BASEMENT

MASSACHUSETTS
INSTITUTE of TECHNOLOGY

HD28
.M414
no.1885-87

OCT 20 1987
LIBRARIES

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

THE "SOFTWARE FACTORY" RECONSIDERED: AN APPROACH TO THE STRATEGIC MANAGEMENT OF ENGINEERING

Michael A. Cusumano
Sloan School of Management
M.I.T.

May 1987 WP #1885-87

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

THE "SOFTWARE FACTORY" RECONSIDERED: AN APPROACH TO THE STRATEGIC MANAGEMENT OF ENGINEERING

Michael A. Cusumano
Sloan School of Management
M.I.T.

May 1987 WP #1885-87

M.I.T. LIBRARIES
OCT 2 0 1987
RECEIVED

Michael A. Cusumano
MIT Sloan School of Management
Working Paper #1885-87

6/1/87
Software Project Paper #1

THE "SOFTWARE FACTORY" RECONSIDERED: AN APPROACH TO THE STRATEGIC MANAGEMENT OF ENGINEERING[1]

Contents:

Introduction
I. Conceptual Framework
II. Software Industries in the U.S. and Japan
III. The Survey
IV. Implications for Engineering Management
Conclusions
Appendix: Data Analysis

INTRODUCTION

This paper examines the question of whether or not companies are choosing to manage a complex engineering activity such as large-scale software development with a range of strategic considerations and organizational as well as technological approaches that corresponds to the spectrum usually associated with "hard" manufacturing, i.e. job shops, batch organizations, and factories exhibiting various degrees of flexibility in product mixes and technologies. There are several interrelated conclusions: (1) This spectrum, including "factory" approaches, is observable in a statistically significant sample of managers at 38 software facilities in the U.S. and Japan. (2) The existence of this spectrum suggests there is a range of beliefs among managers as to how software development should or can be managed; and that there is nothing inherent in software as a technology that prevents some firms from managing the development process in a more disciplined, "factory-like" mannner than others. (3) Japanese firms -- led by

the NEC group and Toshiba, followed by NT&T, Hitachi, and Fujitsu -- are significantly ahead of most U.S. competitors in applying what might be called a disciplined and flexible factory approach -- applying production-management concepts, general-use tools, standardized procedures, effective quality-control techniques -- to large-scale software development.

The research methodology followed was to develop a conceptual model consisting of 23 criteria related to software support tools and technologies as well as to various policies or methodologies for design standardization, documentation, reusability, maintenance, portability, and the like. Surveys were then received from managers at 38 major facilities making two types of products that usually require large amounts of people, time, and tools to develop, and which might provide incentives for managers to seek similaritaries and common components or tools across different projects: operating systems for mainframes or minicomputers ("systems" software); and real-time applications programs, such as for factory control or reservations systems ("applications" software). An underlying assumption was that a persistent worldwide shortage of software engineers and rising demand for computer programs might have convinced some managers to try to rationalize development activities much as their predecessors in other industries have rationalized "hard" manufacturing.

There has been extensive research, as well as trial-and-error development, of software tools and environments over the past decades. The concept of a "software factory" rather than a laboratory as a model for integrating tools and procedures in a systematic, disciplined environment was

first discussed in a 1960s' NATO science conference. It was then attempted (successfully) in Japan beginning in the late 1960s at Hitachi and in the U.S. (with less success) during the mid-1970s at System Development Corporation. Subsequent to Hitachi, most other large Japanese software producers have created disciplined, tool-intensive, centralized facilities for software development, incorporating many process-analysis, production-mangement, and especially quality-control concepts or techniques used in hardware factories. U.S. firms have also made progress in software engineering, but since SDC and the late 1970s, have avoided using the term "factory" to describe their facilities or efforts to improve software productivity.

As a general observation, it seems that the process-management skills Japanese firms have demonstrated in various manufacturing fields, resulting in extrordinarily high levels of productivity and quality control, occur with surprising regularity in their software engineering facilities and far more frequently than in comparable U.S. facilities. This may be a disturbing conclusion to U.S. managers and policy makers who have believed that the U.S. lead over Japan and other countries was insurmountable at least in software. While this paper makes no attempt to deny that the U.S. has a large and talented supply of software programmers, it does suggest that U.S. firms may not be leaders in **implementating software-engineering tools and management techniques**. Moreover, management aspects are becoming increasing important to improving productivity and quality in the software industry as the shortage of programmers and the demand for programs continue. A relative stabilization in the product technology that may be occurring would also make "process innovation" in software more likely,

theoretically, and possible on a practical level.

This paper, and parallel case studies currently in preparation, argue as well that firms can achieve significant standardization and rationalization of the development process even if they cannot or will not standardize their end products as in a mass-production factory. In this sense, the term "software factory" does not imply rigidity. In fact, facilities closest to the proposed factory model either customized products such as applications programs or designed unique products such as operating systems. The factory analogy should thus be thought of as synonomous with a strategic commitment, at the company- or at least facility-level, to providing sufficient scale of people and operations to justify research and development for process technology and techniques; institutionalizing "good" technology and practice; improving process efficiency through teamwork and better inter-group communication; allowing an entire organization to focus more on worker productivity and product quality; and eliminating waste and redundancies due to dysfunctional behavior and the lack of an organizational strategy. Perhaps the most important potential benefit of this approach would be defect control, since data from IBM, TRW, and GTE indicate that fixing bugs in a completed program during operation can cost 100 times that of detecting errors in the design stage.[2]

I. CONCEPTUAL FRAMEWORK

A Spectrum of Manufacturing Strategies and Organizations

Before discussing software, a review of how other products have been developed provides some perspectives on the different strategic and organizational options potentially available to an engineering organization. Throughout history, in a variety of industries, countries, and time periods, product technology for goods such as books, textiles, guns, paper, and automobiles tended to standardize, at least temporarily. As production volumes increased, many companies shifted their focus from product development to process innovation -- including areas such as standardization of components and procedures, specialization and division of labor, task automation -- to evolve from craft-like job-shops or batch operations to large-scale design and production for mass markets. As Chandler has described, American companies were leaders in establishing this new mode of mass production and organization.[3]

Once firms made this transition, the strategic issues job-shop managers faced making one-of-a-kind products -- raising organizational and perhaps technological flexibility, product customization, or quick reaction to individual customers -- tended to give way to concerns such as how to increase volumes, lower unit costs, reduce product variety and complexity, improve standards and process integration, or raise the levels of automation while lowering the skills required of workers. The results of moving in this direction -- which Ford perfected with the Model-T before World War I -- included higher productivity and lower unit costs, and helped make sophisticated goods such as automobiles accessible to large numbers of

consumers. As Abernathy and Utterback postulated in 1975, these benefits appeared to offset the higher quality, greater possibilities for product differentiation, and general flexibility in terms of products, worker tasks, and technology found in job-shop or batch operations.[4]

Authors who wrote prior to the early 1980s accurately pointed out the risks and benefits of basing the strategic management of engineering and manufacturing on this concept of a "product-process matrix" or "life cycle." But they had not fully recognized that, during the 1950s, to accommodate a small but rapidly growing market in Japan, a fourth type of manufacturing (and engineering for manufacturability) model had also appeared. Pioneered most effectively at Toyota, the Japanese managed to combine many of the benefits of mass-production factory environments with the flexibility of batch operations. This approach (gradually imitated by other Japanese firms in a variety of industries) relied upon even stricter process control and standardization of components and procedures than Ford had achieved, but broadened the job specifications of workers, used automation much more selectively, and added an interrelated set of process strategies and techniques (rapid equipment set-up times, low in-process inventory levels and a "just-in-time" manufacturing and delivery discipline, greater use of subcontracting, "total" quality control programs and worker self-inspection.) The entire Toyota system thus served to facilitate production in small rather than large lots (batches). But, in addition to this greater flexibility, the process discipline and innovations it incorporated led to the highest levels of physical productivity among world automakers. The Toyota case also demonstrated clearly that process efficiency was not achieved merely by

technology, such as automated machinery. The most important contributions to production improvements at Toyota appeared to lie in the management policy area -- process analysis, standardization, and worker discipline and cooperation.[5]

As U.S. and European automakers learned from Toyota and other Japanese firms who adopted similar approaches primarily during the 1960s and 1970s, compared to rigid mass-production engineering and manufacturing, a mixture of superior process efficiency and flexibility in the mix of final products made it possible for Japanese companies to compete in the world marketplace on the basis of not simply product differentiation (high reliability and perceived quality) or low cost, but with both. As Porter has suggested, combining product differentiation and cost leadership is, at the same time, very difficult for one firm to achieve but equally difficult for its competitors to overcome.[6]

Also somewhat beyond the vision of the initial product-process life-cyle theorists is a fifth organizational model that appeared in Europe, the U.S. and Japan initially during the 1960s and 1970s, with continued refinements in the 1980s: highly automated and highly flexible manufacturing systems (FMS) that can produce a variety of products quickly and with little or no financial penalty associated with low volumes, once a firm has invested in developing the system. In these environments, which rely heavily on tools and systems such as computer-aided design and manufacturing (CAD/CAM), the ratio of time and money companies once devoted to purely production activities is pushed back into product engineering and

development of systems capable of automating a large number of manufacturing activities. The result is the productivity, control, and quality of a highly automated factory, plus the capability to produce a variety of products or introduce innovative designs or processes quickly and at low cost. This evolution beyond the Ford or "American" style of rigid mass production has been discussed in the work of Piore and Sabel, Jaikumar, Meredith, and others.[7]

The typology of basic production organizations described above, and their accompanying characteristics and tradeoffs, can be summarized as follows:

TYPOLOGY OF BASIC PRODUCTION ORGANIZATIONS

TYPE 1: Job-Shop/Craft Environments
(focus on customized products, general purpose equipment and highly skilled labor, to maximize flexibility in design; full but unsystematic integration of design and production; high margins from unique designs make cost control less a priority)

TYPE 2: Batch-Operations
(focus on low-volume multiple products, but still much customizing and high margins)

TYPE 3: Rigid Mass Production and Engineering
(focus on higher volumes and few products; standardized product technology makes simple, fixed automation and division of de-skilled labor tasks highly economical; specialization can lead to organizational separation of design from manufacturing, with engineering sub-organizations focusing on product development and manufacturing sub-organizations on mass-production)

TYPE 4: Low-Automation Flexible Mass-Production
(focus on process and components standardization and medium to high but controlled volumes, less rigid automation and more flexible equipment (rapid set-up times), and less specialized workers; these facilitate small-lot production, to combine the benefits of product variety associated with batch-operations or job shops with cost and quality controls, and productivity increases, of mass-production

environments)

TYPE 5: High-Level Automation, Flexible Production
(focus on design of a system capable of fully integrating and automating product-engineering and manufacturing functions, so that high-volume and stable product designs (within certain parameters) are no longer important; this captures the benefits of both job-shop flexiblity in product differentiation with the productivity, precision, and standardized high quality and low costs of mass-production-type automation).

Analogies for Engineering

These categorizations seem most applicable to organizations manufacturing "hard" goods such as automobiles, as companies choose different arenas in which to compete, from fully customized products to mass-produced items. Back in the development part of the organization, however, there also appear to be choices of a similar nature. For example, if a customer needs a product, whether it is an automobile, a machine tool, a semiconductor chip, or a software program, there are basically three options: obtain a fully customized product -- from a vendor or an in-house department; obtain a standardized or "packaged" product; obtain a semi-customized product (from a vendor or an in-house department that customizes a purchased standardized product). It follows that vendors should have three corresponding options: 1) sell a customized product; 2) sell a standardized product; 3) customize a semi-standardized product. Companies in the business of making these items then have several options for managing the process of product development as well as manufacturing--and these might very well parallel those of a manufacturing organization, as outlined below.

BASIC STRATEGIES FOR DEVELOPMENT ORGANIZATIONS

STRATEGY 1: Customize each development process for each product ("Job-Shop" Analogy)	**IMPLEMENTATION:** Maximize the capability of the organization to produce a unique product that will capture a high price from at least one customer
STRATEGY 2: Customize some processes and sell more than one of each product ("Batch" Analogy)	**IMPLEMENTATION:** Maximize the capability of the organization to produce a unique product that will capture a large share of the market
STRATEGY 3: Standardize the processes and the products ("Rigid Factory" Analogy)	**IMPLEMENTATION:** Maximize the capability of the organization to produce a product with standard features at the lowest possible price
STRATEGY 4: Standardize the processes but customize the end products, with large-factory efficiency ("Flexible Factory" Analogy)	**IMPLEMENTATION:** Maximize the capability of the organization to produce semi-custom products at a low price through the use of as many standardized procedures and inputs as possible
STRATEGY 5: Customize the products but automate the processes ("FMS" Analogy)	**IMPLEMENTATION:** Maximize the capability of the organization to produce customized products at a low price through the use of highly flexible process techniques and/or automation

Good examples of these strategies can be found in the Japanese auto industry, where Toyota and other local automakers developed an integrated engineering and manufacturing approach that resembles Strategy 4. Extremely low sales in Japan after World War II (total Japanese car and truck production in 1950 equalled merely one day of U.S. output), despite

nearly a dozen producers by 1960, as well as rising demand for a variety of models, encouraged Toyota and then other Japanese automakers to focus first on developing efficient design and production methods for small lots or batches, and then on increasing product variety -- in effect, end-product customization. As volumes increased, companies gained more efficiencies through larger scales of operations, making it possible for Japanese automakers to combine remarkable productivity levels in manufacturing with a variety of products tailored to domestic and export markets. The history of Toyota from around 1948 through the 1970s reveals, however, that the company went through a long series of steps -- first introducing more controls over its engineering and manufacturing outputs, and then gradually adding more product variety, volumes, and automation. But the eventual result was a more flexible factory model -- actually, an integrated engineering and manufacturing philosophy and organization -- than had existed previously:

TOYOTA PRODUCT-PROCESS DEVELOPMENT

1. Low Volume, Low Variety Production
   Organizational Centralization
2. Product Variety Limits
3 Volume Leveling
4. Process Analysis/R&D
5. Establishment of Good Process Techniques
6. Standardization of Procedures
7. Standardization of Components
8. Extension of Worker Job Routines
9. Volume Increases
10 Selective Introduction of Automation/More Process R&D
11. Increase in End-Product Variety
12. More Automation/Process R&D
13. More Product Variety
14. Extension of the System to Other Locations
15. Continual Improvement

Industries such as machine tools, aircraft, specialty motors, construction or agricultural equipment, and defense systems have faced similar problems as Toyota and the Japanese automakers once did. Their design and manufacturing organizations might also be viewed as moving from Strategy 1 or 2 to Strategies 4 and especially 5, bypassing Strategy 3. Customers of these products often require unique or customized features, for different types of applications. As a result, the producers need to design and make products in small batches, with varying degrees of customization. Since development often requires high levels of precision and highly skilled, expensive workers, products may become expensive. This is a major reason why manufacturers of machine tools, aircraft, specialty motors, construction or agricultural equipment, and defense systems have led in the installation or development of FMS technology, which provides the capability of producing products with the precision and efficiency of an automated factory and the customizing capability of a job shop or batch-processing organization.[8]

In semiconductor design, one finds as well a movement from Strategies 1 or 2 to Strategies 4 and 5, as companies have attempted to respond to various customer needs more effectively by exploiting new technological capabilities. For specific applications, design times can range from a few hours for a standard chip to years for fully customized chips, with comparable differences in costs. As demand for application-specific chips has risen, and customer needs have become more predictable, companies have succeeded in developing highly automated equipment to design standardized gate arrays, and then standardized modules or cells, to gain traditional large-scale efficiencies -- in engineering as well as in manufacturing.

Companies then add the necessary customization to their products toward the end of the processing cycle, allowing producers to meet different customer needs at competitive prices.[9]

Another concept usually associated with manufacturing that has brought increasing standardization and rationalization to engineering activities is group technology. The basic idea here is to maximize efficiencies by grouping together similar parts, processes, problems, or tasks, using some sort of classification and coding scheme. Job shops and batch-processing organizations tend to treat each part they design as unique in both design and manufacturing. But, over time, there often appear many similarities in shapes and processes among at least some of the components in production, allowing various companies to report benefits in both cost and quality through increased standardization, rationalization, and simplicity. In general, this seems to be because performing similar activities together avoids wasted time in changing from one activity to another; standardizing closely related activities and focusing only on distinct differences avoids unnecessary duplication of effort and places more effort where it is most important; efficiently storing and retrieving information related to recurring problems reduces search time for the information as well as eliminates the need to solve a problem again.[10]

Large-scale software development is a particularly appropriate case to study the application of manufacturing-type strategies to engineering because, not only is this a growing field providing a product essential to many industries, but it is often considered to involve primarily design rather

than manufacturing activities. This can be seen in a commonly cited breakdown of life-cycle cost components for software: 10% of total costs devoted to requirements definition, specifications, and design; 7% to coding; 15% to testing; and as much as 70% to maintenance, which involves a repetition of earlier steps in order to produce an enhanced or corrected design.[11] Moreover, while software has been evolving from a highly-skilled, even artistic or craft-like activity into a discipline containing many attributes of scientifically and mathematically-based engineering practices, organizational and strategic rationalization seems to have proceeded slowly; and demand for applications software in particular has continued to outpace the capacity of companies to supply products. It seems well worth determining why and how some firms have decided that software engineering need not forever remain in a job-shop or, at best, a batch-mode of operational management and efficiency.

II. THE SOFTWARE INDUSTRIES IN THE U.S. AND JAPAN

Market Comparisons

If general tendencies are different for firms in different countries, differences in industry structures and markets may be key to the explanation. The software industries and markets in Japan and the U.S. do exhibit some significant differences, to which firms might be reacting in determining management strategies for software product and process development.

Both the U.S. and Japan had in common rapid market growth combined with industry shortages of engineers. One U.S. estimate claimed that software demand was growing at 20 to 30% annually while the supply of programmers was increasing at the rate of merely 3 to 4% per year.[12] U.S. companies needing customized applications software for mainframes typically had a 3- to 4-year backlog.[13] Although some producers have attempted to rationalize their operations along the lines of Strategy 4 or 5, it appears that many U.S. software firms have also attempted to develop a few excellent products and then sell them to large numbers of customers. This can be seen in the fact that about 60% of U.S. software sales in a market valued at $10 billion in 1982 and about $30 billion in the mid-1980s were of standardized or "packaged" programs (Strategy 1 or 2).

The U.S. market was also becoming increasingly biased toward small machines. About 15% of programs by value were for mainframes and office or minicomputers; the rest were sold for personal computers.[14] About 70% of the total market consisted of systems software -- operating systems, database management systems, telecommunications monitors, translators, utilities -- and the rest applications software, ranging from spreadsheets for personal computers to simulation packages for supercomputers.[15] Another trend was the increasing power of personal computers, which in 1987 were exhibiting similar speed and memory capabilities as mainframes of only a few years earlier. This was increasing the size and complexity of programs even for small machines and decreasing the historical borders of classifications such as "personal," "mini," and "mainframe," as well as forcing software firms to write programs for a range of machines, further exacerbating the shortage

of software engineers.[16]

Japanese demand for software programmers was increasing at about 26% per year, compared to growth in their supply of about 13% annually, according to a 1986 estimate. This trend was expected to result in a shortage of 600,000 Japanese programmers by 1990.[17] This was less of a shortage than in the U.S., although the Japanese software market was about one-fifth the size of the U.S. market, including the estimated value of systems software bundled with hardware (in the U.S., systems software was usually sold separately). Japanese also wrote the vast majority of their programs for large machines, since personal computers were slower to diffuse. One, perhaps related effect, was that Japanese buyers continued to demand unique products in high numbers. In fact, 95% of Japanese software sales were customized programs almost exclusively for mainframes or minicomputers, compared to about 30% in the U.S.[18]

As discussed in the conceptual framework presented earlier in this paper, the shortage of programmers and the demand for customization might have persuaded at least some Japanese applications producers to standardize or rationalize their development operations while continuing to produce differentiated products for their customers. This would involve opting for either Strategy 4 or 5. The techniques and technology to accomplish this rationalization have been under development in both Japan and the U.S. since the 1960s, when practicioners found that systematic methods of analysis as well as concepts or techniques borrowed at least in part from science and mathematics were useful in software environments.[19] There was

surprising wide agreement in the academic and business communities by the mid-1980s on what constituted "good" practices and tools. But disagreements remained regarding the nature of software development (art vs. science vs. engineering).[20] And, as the survey revealed, there are fairly wide differences among firms in the degree to which similar practices and tools were being emphasized.

The Field of Software Engineering

Led by engineers at firms such as IBM and TRW in the U.S., the emerging discipline of "software engineering" has produced technologies and methods such as high-level languages; automated support tools for design, coding, documentation and testing; workbenches; prototyping techniques; program libraries; quality metrics; designs relying on structured programming and data or procedure abstractions; and techniques for project management and control.[21] Yet, for no doubt a variety of reasons, related or unrelated to the nature of software technology or debates about its characteristics, software management does not seem to have proceeded as rapidly or confidently as tool and method technology that primarily facilities the work of individual engineers or small groups.[22]

In "hard" industries such as automobiles, machine tools, or even semiconductors, there was a tendency of companies to shift more of their emphasis to rationalizing or innovating in process technology once the product technologies stabilized. This stabilization of product technology allowed new production organizations, such as the mass-production factory,

to emerge and dominate an industry, at least until market demands changed and competitors discovered superior modes of production to meet the new requirements.[23] A key question engineering organizations might ask is whether their product technologies exhibit enough stabilization to consider further rationalization of the development process.

With regard to software, one might argue that product technology is stabilizing While operating systems and applications like banking programs have changed in complexity as, say, cars have, distinct product types have appeared and become rather standardized in terms of functions and customer expectations. The huge share of the U.S. software market (nearly 60%) devoted to non-customized "packaged" software indicates clearly that this is taking place.[24] Furthermore, the survey reveals that many managers in both the U.S. and Japan, especially in applications facilities, found much of the software they were developing as reusable in a recent sample year and some reused large percentages of code (see table below).[25]

PERCENT OF CODE CONSIDERED REUSABLE AND REUSED

Notes: Asterisk (*) indicates author's estimate based on averaging of multiple company statistics.

% Rework Allowed refers to the maximum percentage of code a manager would allow to be reworked before the module would no longer be classified as reused.

	% Reusable	% Reused	% Rework Allowed
Applications Producers			
Japanese	15	8	50/60
Japanese	--	55	--
Japanese	35	--	--
Japanese	40	20	40
US	30	10	50
US	5	15	40/50
US	20	15	--
US	50*	10	25
US	5	5	--
US	15	10	--
US	1	1	--
US	10	10	50
US	25	50	10
US	75	18	10
Systems Producers			
Japanese	--	35	--
Japanese	85	50*	--
US	40	40	25
US	15	10	25

Source: Survey responses from managers at Fujitsu, Toshiba, Hitachi, NT&T, Nippon Electronics Development, Unisys/SDC, EDS, TRW, Boeing Aerospace (systems and applications), Honeywell, Computervision, Martin Marietta (Denver and Maryland), Hughes Aircraft, Unisys/Sperry, Data General, and IBM.

"Factory" Approaches in the U.S. and Japan

At the operational level, one way to rationalize a complex processing activity activity is by centralizing the locus of activities to gain certain economies of scale or scope; providing R&D for the relevant process technologies; and integrating technologies with policies defining tools, methods, or procedures in a way to enhance organizational productivity.[26] Historically, production organizations with these characteristics emerged during the 18th and 19th centuries in Britain, Europe and the United States in the textile industry and then gun-making. They were called "factories," and they institutionalized several innovations intended to raise worker productivity and lower unit costs: integration of various production processes in a large, centralized facility; close physical coordination of the flow of each process; division and specialization of labor; mechanization of tasks; and rigid accounting controls.[27]

The term "factory" for software is somewhat of a misnomer, in that software development includes planning, engineering (design), production (coding and testing), and maintenance (redesign) activities; and many software facilities are engaged in customizing products or developing unique products such as operating systems or data base systems. The first American company to attempt to implement a factory model for customized applications software was the System Development Corporation (SDC), then a Burrough's subsidiary (now part of Unisys) that had been one of the contractors of the SAGE air defense system when it was part of the Rand Corporation. SDC developed real-time software primarily for government contracts, and in the mid-1970s put together an integrated set of tools

(program library, project databases, on-line interfaces, and automated support systems for verification, documentation, etc.) that were supposed to work in conjunction with a set of standardized procedures and management policies for program design and implementation. This system SDC copyrighted under the name "The Software Factory."

SDC engineers were particularly interested in dealing with five common problems in software development; analogies to these appeared in other engineering and manufacturing activities as well: (1) Lack of discipline and repeatability or standardized approaches to the development process, with the result that SDC was continually reinventing products and processes, and not becoming as proficient at development or project control as managers wanted. (2) Lack of an effective way to visualize and control the production process, as well as to measure before the project was completed how well code implemented a design. 3) Difficulty in accurately specifying performance requirements before detailed design and coding, and recurrence of disagreements on the meaning of certain requirements, or changes demanded by the customer. 4) Lack of standardized design, management, and verification tools, making it necessary to reinvent these from project to project. 5) Little capability to reuse components, despite the fact that many application areas used similar logic and managers believed that extensive use of off-the-shelf software modules would significantly shorten the time required for software development.[28]

There were serious implementation problems with the factory concept as applied at SDC. As discussed in another paper, project managers did not

like giving up control of development efforts to a centralized facility; there was not always a steady flow of similar work into the factory; programmers seemed to dislike the rigid environment and reusing code from a central source. Overall, it seems that management attempted to impose the factory infrastructure of tools and methods on both managers and programmers without preparing both personnel and the workflow to the new system. In any case, SDC gradually abandoned the factory experiment by the late 1970s, although it has continued to use many of the factory procedures and at least one of the tools.[29]

In contrast, the factory concept has found more of a following in Japan, beginning with Hitachi's opening of the world's first facility called a software factory in 1969 and then Toshiba's in 1977.[30] Key managers responsible for software development at these two firms as well as at NEC have also indicated they were influenced by SDC attempts to discipline software development as early as the 1960s.[31] This does not mean that the Japanese were choosing to develop radically different technology; in fact, a recent survey comparing U.S. and Japanese software practices and various other articles maintain that the type of technology for software development is quite similar in Japan and the U.S. The difference this survey found was that Japanese firms appeared to be developing and using recommended tools and methods more systematically than their U.S. counterparts.[32]

In general, however, as in the manufacture of automobiles and other products, the Japanese software producers appear to have set high standards for process analysis and defect control, as well as for general production

management and productivity. Reports on recent developments in software engineering at large Japanese firms seem to fall into four categories, all of which writers have assumed represent good practice and a departure from U.S. norms:[33]

-- attempts to exploit Japanese traditions of teamwork, discipline, and individual attention to quality by developing software tools and planning or reporting systems that facilitate group programming and a teamwork methodology throughout the software life cycle.

-- quality control techniques designed to catch bugs early, before they become difficult to fix.

-- national and company efforts to improve software quality and productivity through reusability of software modules and automation of software production (code generation).

-- construction of large, factory-like facilities to integrate the entire process of software development.[34]

Government and private surveys have also expressed concern about the impact of these developments on what many Americans have felt was an unassailable U.S. lead in software engineering skills. For example, a U.S. Department of Commerce report asserted in 1984 that the Japanese were more "disciplined" and thus were placing more emphasis on developing tools and "factories," while U.S. programmers suffered from viewing software development too much like a "craft":

> The Japanese have...made impressive gains in the development of software tools and have encouraged their widespread use within their software factories to boost productivity...By contrast, while the United States is developing software engineering technology, the use of tools in U.S. firms is quite limited... Many U.S. software companies consider programming a craft and believe the future strength of the industry lies in its creativity rather than a disciplined approach to software development as do the Japanese."[35]

A 1985 article in the **Electronic Engineering Times** similarly claimed the Japanese were more effectively utilizing team or group approaches, as well as developing unique team-oriented software tools, while Americans were were becoming overly dependent on small groups and highly skilled individuals:

> "[T]he approach to software technology taken by major developers in Japan, such as NEC, Fujitsu Ltd, and Hitachi Ltd., universally strive to harness that tradition of excellent teamwork... Each of these developers has automated versions of planning and reporting systems that enforce a strict teamwork methodology through the complete life cycle of a computer program -- from planning to design to maintenance, and without coding, since high-level language-source codes are automatically produced from the design documents.
>
> ... Until now, the Japanese have been hampered in their software development efforts by a lack of team-oriented tools. The tools borrowed from the United States simply do not fit the Japanese culture because they put too much control in too few hands.
>
> In America, industrial software development is generally done in groups that are as small as possible to minimize the communication problems among people. That makes the knowledge of each individual programmer a critical factor to the success of any software-development project. But...that is just not tolerable in the Japanese culture.
>
> As a consequence, the Japanese have had to perform basic research into software tools that can be wielded by many hands at once. Nobody else was going to develop group-programming tools for them."[36]

Another trend perhaps was the potential for rapid dissemination of software tools and expertise in Japan, because industry activity was so concentrated at the top four computer manufacturers -- Fujitsu, NEC, Hitachi, and Toshiba. The basic industry consisted of 450 companies registered as members of the Japan Information Service Industry Association.[37] But most of these were extremely small in terms of employees and revenues, although 17 companies producing software as their

major business had 1000 or more employees in 1983.[38] An estimated 50% of all the software Japan developed came included (bundled) with large and medium-size computer hardware manufactured primarily by the top four computer companies. NT&T and Mitsubishi Electric were the only other significant Japanese producers of systems software, excluding subsidiaries of U.S. firms.[39]

Not only did NEC, Fujitsu, and Hitachi rank one, two, and three in software revenues in Japan.[40] These three companies have also transferred much of their technology and development tools to subsidiaries, which ranged in size from a few dozen employees to nearly 2500. Fujitsu, for example, had 52 software subsidiaries in 1986, Hitachi 24, and NEC 21.[41] Two of the top five independent software producers in Japan, Nippon Business Consultants (#1 in sales in fiscal 1983) and Hitachi Software Engineering (#5), were Hitachi subsidiaries.[42]

Japanese computer companies, like Japanese firms in other industries, despite some similarities, were still not monolithic in their approaches to software development.[43] While Hitachi and Toshiba claimed publicly to operate "software factories," NEC and Fujitsu did not. There was not even a consensus at NEC and Fujitsu that a "factory" was an appropriate model for software development; at least some managers at these firms felt software was essentially a "design" activity unsuitable for "mass production."[44] Fujitsu also called its large-scale applications facility an "Information Processing Systems Laboratory." Nevertheless, NEC and Fujitsu still produced software centralized, tool-intensive environments housed in

large facilities officially designated as "factories." And, like Hitachi and Toshiba, both companies appeared to be rigorously applying to software hardware-type statistical quality control, inspection practices, and quality circles.[45]

Since the dissolution of SDC's Software Factory, SDC and other American firms appear to have preferred designations such as "laboratory" or "systems development center," or no label at all, to designate their software organizations.[46] Yet many U.S. companies, such as TRW, IBM, Boeing, appear to have gone considerably beyond the SDC Software Factory experiment in studying tools and methods for design and testing, as well as programming environments and managerial aspects of large-scale software production.[47] Yet to be answered, however, is the question of what degree of integration and standardization among people, systems, functions, tools, methods, inputs, and the like seems appropriate to distinguish a "software factory" from simply a large facility housing discrete groups of engineers. It seems plausible as well that at least some U.S. firms have recogized a need to rationalize software development and today operate in a manner sufficiently integrated, standardized, and strategic so that they might as well be called "flexible factories," especially is measured by the ideals SDC originally set out to implement.

III. THE SURVEY

Methodology

The survey was designed to determine where U.S. and Japanese companies stand in relation to a set of criteria suggested by the SDC Software Factory experiment. In particular, it was thought that mapping companies along the spectrum that emerged would make it possible to examine if several hypotheses about software organizations and the U.S.-Japan comparison, described later in this section, were true or not.

Major producers of large-scale systems and applications software in Japan and the U.S. were identified through public literature and discussions with industry experts. All the Japanese firms contacted filled out the survey; the vast majority of U.S. firms contacted also decided to participate. To improve the comparability of responses, the survey was sent to managers of (1) facilities producing operating systems or network systems software for mainframes or minicomputers; and (2) facilities producing real-time applications or control programs for mainframes or minicomputers. Two managers at each type of facility either responsible for overall software engineering management or with sufficient experience to present an overview of practices for the entire facility, were asked to respond. About half the companies returned two completed surveys for each type of facility; these answers were averaged. Questions and answers were often clarified through discussions with the respondents. While the Japanese sample size is small, the surveyed firms account for the vast majority of software written and sold in Japan, as indicated in the previous section.

The survey criteria, and answers key, were as follows:

SURVEY ANSWERS KEY:
4 = CAPABILITY OR POLICY IS FULLY USED OR ENFORCED
3 = CAPABILITY OR POLICY IS FREQUENTLY USED OR ENFORCED
2 = CAPABILITY OR POLICY IS SOMETIMES USED OR ENFORCED
1 = CAPABILITY OR POLICY IS SELDOM USED OR ENFORCED
0 = CAPABILITY OR POLICY IS NOT USED

I. TECHNOLOGICAL INFRASTRUCTURE

A. Centralization of development for a distinct software product family (such as an operating system like IBM's VM or DEC's VAX/VMS) in a single location or directly linked sites operating as an integrated unit, rather than decentralizing development in independent sites.

B. A uniform set of specification, design, coding, testing, and documentation procedures used among project groups within a centralized facility or across different sites working on the same product family to facilitate standardization of practices and/or division of labor for programming tasks and related activities.

C. A centralized program library system to store modules and documentation.

D. A central production or development data base connecting programming groups working on a single product family to track information on milestones, task completion, resources, and system components, to facilitate overall project control and to serve as a data source for statistics on programmer productivity, costs, scheduling accuracy, etc.

E. Project data bases standardized for all groups working on the same product components, to support consistency in building of program modules, configuration management, documentation, maintenance, and potential reusability of code.

F. A specific group or groups designated to develop and disseminate methodologies and tools to automate tasks such as requirements specification and design, coding, documentation, system testing and debugging, as well as to facilitate standardization of practices and division of labor, and effective managerial control over all programming activities.

G. A system interface providing the capability to link support tools, project data bases, the centralized production data base and program libraries.

H. Automated or semi-automated integration of applicable data from

support tools and development data bases with management control systems (project data bases and the central production data base), for each phase of program development; and the utilization of this capability to facilitate budgeting, forecasting, maintenance, and overall life-cyle cost control on current and future projects.

II. METHODOLOGY & POLICY INFRASTRUCTURE

A. Use of a standardized design language

B. Use of a standardized module-specification language

C. Use of a standardized coding language

D. Emphasis on high-level abstraction (data-type or procedure abstraction; object rather than variable orientation)

E. Planning for maintainability at the module-design level

F. Planning for reusability at the module-design level

G. Planning for portability at the module-design level

H. Monitoring of how much code is being reused

I. "Layering" of reused modules from the program library, along with newly written code, to create new programs

J. Cataloging for the program library of common functional modules (e.g. a date verification routine)

K. Cataloging for the program library of data abstraction modules (e.g. table or linked-list managers)

L. Writing of documentation to accompany modules placed in the program library

M. Requirement that, if changes are made in the code of a module in the program library, the documentation must also be changed

N. Formal management promotion (beyond the discretion of individual project managers) that new code be written in modular form with the intention that modules (in addition to common subroutines) will then serve as reusable "units of production" in future projects

O. Formal management promotion (beyond the discretion of individual project managers) that, if a module designed to perform a specific function (in addition to common subroutines) is in the program library system, rather than duplicating such a module, it should be reused

ADDITIONAL INFORMATION QUESTIONS
(These were confidential; some results are reported in tables and the notes, without revealing company names.)

Data and Hypotheses

The basic data from the surveys is summarized below, followed by the hypotheses tested and the conclusions from the data analysis. The number of NEC facilities included in the sample is large but also reflects its role in the Japanese software market. NEC was the largest producer of software among the Japanese computer manufacturers and had approximately 50% more software revenues than Fujitsu (#2), and three times that of Hitachi (#3) in 1985.

SURVEY RESULTS: DATA SUMMARY TABLE

I = Technological Infrastructure (32=100%)
II = Policy/Methodology Infrastructure (60=100%)
III = Total Factory Model (92=100%)
@ indicates two responses and averaged or joint responses.

n = 38

Applications Means	**69**	**62**	**65**
Japanese (*)	71	72	72
U.S.	67	55	60
Systems Means	**75**	**68**	**70**
Japanese (*)	82	78	80
U.S.	69	57	61
OVERALL MEANS	**71**	**64**	**67**
JAPANESE (*)	76	75	75
U.S.	68	56	60

COMPANY/FACILITY	I	II	III
Applications			
*NEC@	89%	89%	89%
*Toshiba Software Factory	84	87	86
*NEC Information Service	81	88	86
*NT&T Comm. & Info. Proc. Lab.@	81	77	78
*Hitachi Omori Works	78	73	75
*Fujitsu Info. Proc. Sys. Lab.@	77	73	75
*Nippon Systemware	72	70	71
*Nippon Business Consultant	56	67	63
*Hitachi Software Engineering@	53	50	51
*Nippon Electronics Development	41	48	46
TRW	97	83	88
Unisys/Sperry@	91	72	78
Unisys/SDC	72	77	75
Control Data@	84	67	73
Martin Marietta/Maryland	59	76	70
Hughes Aircraft	83	63	70
Boeing Aerospace@	84	53	64
AT&T Bell Labs	72	58	63
Cullinet	64	59	61
Martin Marietta/Denver	69	43	52
Electronic Data Systems@	61	43	49
Honeywell/Defense Systems@	44	42	42
Draper Laboratories@	34	17	23
Computervision@	28	25	26
Systems			
*NEC/Switching Systems	98%	99%	99%
*NEC/Operating Systems	92	92	92
*Toshiba Software Factory	84	87	86
*NEC Software	84	87	86
*Hitachi Software Works@	78	70	73
*Fujitsu Numazu Factory@	77	68	71
*NT&T Comm. & Info. Proc. Lab.@	58	48	51
Control Data@	78	67	71
IBM Endicott	78	62	67
Data General Westboro & N.C.	61	63	62
Boeing Aerospace@	77	53	61
Unisys/Sperry@	61	62	61
IBM Raleigh	84	43	58
DEC (VMS)	41	48	46

RANKINGS: TECHNOLOGY/FACILITY INFRASTRUCTURE

8 Questions; 32=100%

Key:
A.J. = Applications Japan
A.U. = Applications U.S.
S.J. = Systems Japan
S.U. = Systems U.S.
* = Japanese firms

	COMPANY/FACILITY	%	
S.J	*NEC/Switching Systems	98	Flexible
A.U.	TRW	97	Factory
S.J.	*NEC/Operating Systems	92	Approach
A.U.	Unisys/Sperry	91	
A.J.	*NEC	89	
A.J.	*Toshiba Software Factory	84	
S.J.	*Toshiba Software Factory	84	
A.U.	Boeing Aerospace	84	
S.U.	IBM Raleigh	84	
S.J.	*NEC Software	84	
A.U.	Hughes Aircraft	83	
A.J.	*NEC Information Service	81	
A.J.	*NT&T Comm. & Info. Proc. Lab.	81	
A.J.	*Hitachi Omori Works	78	
S.U.	IBM Endicott	78	
A.U.	Control Data	78	
A.J.	*Fujitsu Info. Proc. Sys. Lab	77	
S.U.	Control Data	77	
S.J.	*Hitachi Software Works	78	
S.J.	*Fujitsu Numazu Factory	77	
S.U.	Boeing Aerospace	77	
A.J.	*Nippon Systemware	72	
A.U.	AT&T Bell Labs	72	
A.U.	Unisys/SDC	72	
A.U.	Martin Marietta/Denver	69	
A.U.	Cullinet	64	
S.U.	Data General Westboro & N.C.	61	
A.U.	Electronic Data Systems	61	
S.U.	Unisys/Sperry	61	
A.U.	Martin Marietta/Maryland	59	
S.J.	*NT&T Comm. & Info. Proc. Lab.	58	
A.J.	*Nippon Business Consultant	56	
A.J.	*Hitachi Software Engineering	53	
A.U.	Honeywell/Defense Systems	44	
S.U.	DEC (VMS)	41	
A.J	*Nippon Electronics Development	41	
A.U.	Draper Laboratories	34	
A.U.	Computervision	28	Job Shop

RANKINGS: POLICY/METHODOLOGY INFRASTRUCTURE

15 Questions, 60=100%

	COMPANY/FACILITY	%	
S.J.	*NEC/Switching Systems	99	Flexible
S.J.	*NEC/Operating Systems	90	Factory
A.J.	*NEC	89	Approach
A.J.	*NEC Information Service	88	
A.J.	*Toshiba Software Factory	87	
S.J.	*Toshiba Software Factory	87	
S.J.	*NEC Software	87	
A.U.	TRW	83	
A.U.	Unisys/SDC	77	
A.J.	*NT&T Comm. & Info. Proc. Lab.	77	
A.U.	Martin Marietta/Maryland	76	
A.J.	*Fujitsu Info. Proc. Sys. Lab.	73	
A.J.	*Hitachi Omori Works	73	
A.U.	Unisys/Sperry	72	
S.J.	*Hitachi Software Works	70	
A.J.	*Nippon Systemware	70	
S.J.	*Fujitsu Numazu Factory	68	
A.J.	*Nippon Business Consultant	67	
A.U.	Control Data	67	
S.U.	Control Data	67	
S.U.	Data General Westboro & N.C.	63	
A.U.	Hughes Aircraft	63	
S.U.	IBM Endicott	62	
S.U.	Unisys/Sperry	62	
A.U.	Cullinet	59	
A.U.	AT&T Bell Labs	58	
S.U.	Boeing Aerospace	53	
A.U.	Boeing Aerospace	53	
A.J.	*Hitachi Software Engineering	50	
S.J.	*NT&T Comm. & Info. Proc. Lab.	48	
S.U.	DEC (VMS)	48	
A.J	*Nippon Electronics Development	48	
S.U.	IBM Raleigh	43	
A.U.	Martin Marietta/Denver	43	
A.U.	Electronic Data Systems	43	
A.U.	Honeywell/Defense Systems	42	
A.U.	Computervision	25	
A.U.	Draper Laboratories	17	Job Shop

RANKINGS: TOTAL FACTORY MODEL

23 Questions, 92=100%

COMPANY/FACILITY		%	
S.J.	*NEC/Switching Systems	99	Flexible
S.J.	*NEC/Operating Systems	91	Factory
A.J.	*NEC	89	Approach
A.U.	TRW	88	
A.J.	*NEC Information Service	86	
A.J.	*Toshiba Software Factory	86	
S.J.	*Toshiba Software Factory	86	
S.J.	*NEC Software	86	
A.J.	*NT&T Comm. & Info. Proc. Lab.	78	
A.U.	Unisys/Sperry	78	
A.U.	Unisys/SDC	75	
A.J.	*Hitachi Omori Works	75	
A.J.	*Fujitsu Info. Proc. Sys. Lab.	75	
S.J.	*Hitachi Software Works	73	
A.U.	Control Data	71	
S.J.	*Fujitsu Numazu Factory	71	
A.J.	*Nippon Systemware	71	
A.U.	Martin Marietta/Maryland	70	
S.U.	Control Data	70	
A.U.	Hughes Aircraft	70	
S.U.	IBM Endicott	67	
A.U.	Boeing Aerospace	64	
A.J.	*Nippon Business Consultant	63	
A.U.	AT&T Bell Labs	63	
S.U.	Data General Westboro & N.C.	62	
S.U.	Boeing Aerospace	61	
A.U.	Cullinet	61	
S.U.	Unisys/Sperry	61	
S.U.	IBM Raleigh	58	
A.U.	Martin Marietta/Denver	52	
S.J.	*NT&T Comm. & Info. Proc. Lab.	51	
A.J.	*Hitachi Software Engineering	51	
A.U.	Electronic Data Systems	49	
S.U.	DEC (VMS)	46	
A.J	*Nippon Electronics Development	46	
A.U.	Honeywell/Defense Systems	42	
A.U.	Computervision	26	
A.U.	Draper Laboratories	23	Job Shop

PROJECT CONTROL AND QUALITY ANALYSIS[48]

KEY:

A = Avg Bugs Reported By Users Per 1000 Lines of Debugged Code
B = Average Percent of Projects Late in a Recent Year
C = Average Percent of Projects Finished within 5% of Budget
F = % of Total Factory Model Criteria
(HH = 80% or above; H = 70 to 79%; M = 58 to 69%; L = below 58%)
* = Author's Estimate

FACILITY	A	B	C	F
A.J.	0.13	--	--	HH
A.U.	--	5	95	HH
A.J.	0.15	0	50	H
A.J.	0.04	0	--	H
A.J.	0.7	--	100	H
S.J.	0.02	14	--	H
S.J.	0.01*	5	50	H
A.U.	--	75	10	H
A.U.	0.1	0	100	H
A.U.	1.0	50	80	H
A.U.	2.2	85	5	H
S.U.	0.4	20	20	H
AVG.	0.48	25%	57%	
A.U.	15.0	50	50	M
A.U.	--	20	85	M
S.U.	--	50	50	M
S.U.	--	--	85	M
S.U.	20.0	50	50	M
AVG.	17.5	43%	64%	
S.J.	0.2	7	--	L
A.J.	0.07	--	--	L
A.J.	1.0	--	0	L
A.U.	10.0	15	90	L
A.U.	17.5	28	80	L
A.U.	--	50	20	L
A.U.	--	50	0	L
A.U.	--	90	0	L
AVG.	5.75	40%	32%	

COUNTRY AVERAGES				
Japan	0.26	5%	50%	H
U.S.	8.3	43%	51%	M

Source: Companies responding were Toshiba, NT&T, Fujitsu, Nippon Electronics Development, Hitachi Software Engineering, Hitachi, Unisys/SDC and Unisys/Sperry, EDS, TRW, Boeing, Honeywell, Computervision, Martin Marietta (Denver and Md.), Hughes Aircraft, Cullinet, Draper, Data General, IBM, Boeing, Control Data.

REUSBILITY OF CODE AND THE FACTORY MODEL

% Factory = % of Total Factory Model Criteria
(HH = 80% or above; H = 70 to 79%; M = 58 to 69%; L = below 58%)

Facility	% Reused	% Factory	
A.J.	55	HH	
A.U.	50	H	
S.J.	35	H	
A.U.	18	H	
A.U.	15	HH	
A.U.	10	H	
A.U.	10	H	
A.J.	8	H	HH & H AVG. = 25%
S.U.	40	M	
A.U.	10	M	
S.U.	10	M	
S.U.	0	M	M AVG. = 15%
S.J.	50	L	
A.J.	20	L	
A.U.	15	L	
A.U.	10	L	
A.U.	5	L	
A.U.	1	L	L AVG. = 17%

OTHER AVERAGES:		
Japan	34	H
U.S.	15	M
Applications	13	M
Systems	27	M

Source: Survey responses from managers at Fujitsu, Toshiba, Hitachi, NT&T, Nippon Electronics Development, Unisys/SDC, EDS, TRW, Boeing Aerospace (systems and applications), Honeywell, Computervision, Martin Marietta (Denver and Maryland), Hughes Aircraft, Unisys/Sperry, Data General, and IBM.

HYPOTHESIS 1: The scores of all the facilities in the sample should follow a normal distribution.

Reasoning: If engineering organizations follow a similar spectrum of strategies as found in manufacturing organizations, there should be an observable spectrum of software facilities roughly corresponding to job shops on the one end, and flexible factories on the other, with most firms falling in between in a normal distribution.

Result: Accept hypothesis (kurtosis of 0.31 for the total model).

HYPOTHESIS 2: There should be no significant difference in the average scores for technology infrastructure between Japanese and U.S. facilities.

Reasoning: Although the literature suggests Japanese firms have been more systematic in their use of software tools, it also suggests that general development of software technology has been largely similar in Japan and the U.S.

Result: Accept hypothesis.

HYPOTHESIS 3: The average Japanese score for policy/methodology infrastructure as well as the total score should be significantly higher than the comparable U.S. scores.

Reasoning: The general literature and information from Japanese firms suggests at least the larger Japanese computer manufacturers have been more aggressive in pursuing a disciplined, even a "factory" approach to software development.

Result: Accept hypothesis.

HYPOTHESIS 4: Facilities farther toward the total factory model should exhibit control characteristics one might expect to find in a large factory-type organization: (A) fewer defects (bugs) reported by users; (B) more precise project scheduling; and (C) tighter project cost control.[49]

Result: (A) Accept hypothesis.
(B) Accept hypothesis.

(C) Reject hypothesis.

HYPOTHESIS 5: Facilities that exhibit another characteristic one might expect to find in a large factory-type organization--greater use of standardized components, i.e. higher rates of reusability of code -- should also score high on the total factory criteria.

Result: Accept Hypothesis (Tentatively)

Comments

Formal hypothesis tests for 1, 2, and 3, and other data summaries, can be found in the Appendices.

Facilities or firms in the survey formed a statistically significant normal distribution, moving upward from what has been equated to a job-shop approach to a flexible-factory approach. Without Draper and Computervision, the sample formed a near perfect normal distribution. With or without the outliers, U.S. producers tended to be toward the middle (batch mode) or bottom (job shop), and Japanese toward the top (flexible factory).

In fact, despite the smaller size of the Japanese group, Japanese companies accounted for 13 of the top 17 facilities ranked by the total factory model criteria. NEC, which has not publicly adopted the "software factory" concept, and its subsidiaries consistently led in the rankings. The company that follows the NEC facilities, however, is a U.S. firm -- TRW.

Also noteworthy is Toshiba, which develops both applications software and mini-computer operating systems in the same facility. Since NEC, TRW, Toshiba and other firms in the applications area customize their end products, they, as well as systems producers, which develop unique operating systems and other basic software, seem to fit the "flexbile factory" model best. Another US applications producer making similar types of applications programs to TRW -- Draper Laboratories -- scores lowest on all criteria. Draper and Computervision were "outliers" in the sample but seem to fit the highly creative job-shop model best. Both companies, especially Draper, are known as producers of innovative software technology.

Only 4 of 17 Japanese facilities scored below the median for the total factory criteria. Three of these -- Nippon Business Consultant, Hitachi Software Engineering, and Nippon Electronics Development -- to a large extent served as manpower suppliers to their parent firms (Hitachi in the case of the first two and NEC in the case of the last one) and have not focused on developing their own centralized program libraries, tool groups, etc., but tend to use those of their parent firms.

The greatest deviations (more than 1.2 points) among firms were on the following criteria: I.G (system interface linking tools and data bases); II. B (standardized module specification language), H (monitoring of reused code), J and K (cataloging modules for the program library). Japanese firms tended to score higher on II. F (planning for reusability) and II. O (formal management promotion that modules be reused from the program library), as well as II. J and K (see the Appendix for data summary). These were all

policy or methodology measures, except for the tool-database interface.

With regard to hypotheses related to performance, the data is sketchy for many firms, either because they did not track these measures or would not report them in the survey. The data available shows that some firms scoring low on the factory model, especially Japanese firms, still did well in the control and quality measures. And some firms scoring high on the model did not do so well in the control and quality measures. On average, however, firms closer to the factory model scored better in quality (bug control) and schedule control. This was not so obvious for cost control, however, leading to the rejection of Hypothesis 4c. The explanation may be that many firms place a higher emphasis on completing a project on schedule and with as few defects as possible, and they are willing to spend beyond the budget to accomplish this, even in a "factory" environment.

On the other hand, additional information questions in the survey indicate that performance criteria managers used were very similar in the U.S. and Japanese firms, and they frequently mentioned meeting cost, quality (reliability, specifications), and delivery targets (see Appendix). There was somewhat more emphasis on "productivity" in the Japanese responses, but the answers show that practically all the responding managers valued the basic control measures one would expect to have in a disciplined factory or engineering environment.

With regard to reusability, slightly less than half the firms or facilities in the survey provided data on this; many admitted they did not track this

measure. The data available suggests reuse is higher on average at firms scoring 70% or above in the survey. The hypothesis that firms with high reuse rates were also scoring high on the factory critieria seems acceptable, although the evidence does not seem that strong, and the sample of reporting firms is small. The five firms or facilities that reported reuse rates of 35% or above averaged 73% on the technology criteria, 66% on the policy criteria, and 68% overall, scores only 1 or 2 percentage points above the sample means for these categories. Furthermore, not all firms scoring high on the criteria were reusing lots of code, and not all firms scoring low were poor in reusability. One firm with a very high rate (50%) scored in the low range in the survey criteria, and another with 40% reusability scored in the medium range. And one firm scoring over 80% in the criteria reported a relatively low reusability rate (15%).

A disciplined, centralized environment may provide the technology and methods to facilitate reusability, but managers still need to develop a strategic commitment and program to optimize for this objective. High reuse of code may also simply require that a facility focus its product lines.[50] In addition, there are other ways of thinking about reusability other than simple blocks of code. Toshiba, for example, keeps "reuse" figures for whole program parts, as well as skeletons of modules, utility subsystems, and support tools.[51] The topic of reusability will be treated in the case studies, but is apparently a complex subject. It is also a topic dealt with extensively in software engineering literature.[52]

IV: IMPLICATIONS FOR ENGINEERING MANAGEMENT

The factory model seems to bring benefits in quality control and scheduling control. There is also considerable evidence that, through reusability, enormous improvements in nominal productivity (lines of code produced per programmer in a given time period) are also possible. Why might this type of environment lead to improvements in engineering management and performance? At the same time, how much might a commitment to a factory rather than a laboratory model constrain the "flexibility" of a firm, in terms of meeting different or changing customer needs or accommodating technological change?

Scale to Justify Process R&D

Numerous tools and new methods or techniques have greatly improved productivity in software, as well as in probably every other field of engineering or manufacturing. Factory-type centralization, whether it is geographic or electronic, and the guarantee that large numbers of programmers will use the results of this investment, may be important for a firm to justify the costs of continuous tool or method development and funding of software-engineering groups. In NEC, Hitachi, and Toshiba, for example, specific engineering departments have been formed to keep track of the latest developments in software engineering, and experiment with their own ideas and tools. They also have the authority and responsibility to introduce what they decide are "good practices" into facilities or subsidiaries housing thousands of programmers.

The degree of centralization of firms in the sample varied; NEC, for example, was more decentralized than Hitachi, though more standardized. In all, 17 of the 38 facilities in the survey responded with answers of 4.0 or 3.5 to question I.F (emphasis on having a specific software engineering tools and practices group). Since only seven were from the U.S. (Boeing, TRW, Unisys/SDC, Hughes, IBM, and Control Data), 60% of the Japanese respondees and only one third of the U.S. respondees fully used this type of a group.

Institutionalizing Good Technology and Practice

It is not clear how centralized software development should be to meet different customer needs or to prevent reaching a scale where the numbers of people become too large to manage. In general, however, the basic tools and procedures proposed in the survey criteria represent recommended "good practice" in the software industry since at least SDC and the mid-1970s Software Factory experiment. In a practical sense, one might thus view the factory analogy as a strategy moving away from the undisciplined atmosphere of a job shop or laboratory; and as a mechanism for management to introduce and then require the use of "good" technologies and practices throughout the appropriate parts of an organization, rather than leave technical choices, tool development, and the like to individual discretion. There may be some time lag in adopting the latest methods or tools, for design, testing, reusability, maintenance, or whatever. But companies with formal commitments to understand and then disseminate new process technology may find that the knowledge and support levels of their average

engineers, over time, are higher than in firms without comparable strategic commitments to institutionalizing good technology and practices.

There is considerable evidence that factories in operation are promoting skill development with extensive training programs.[53] For example, dividing up tasks or creating reusable and easily maintainable program parts as done in several Japanese facilities require fairly sophisticated programming concepts, such as data or procedure abstraction (otherwise no modules would be reusable), or even "layering" of reused modules with newly written code.[54] Toshiba has made abstraction central to its reusability strategy, after realizing that the higher the level of abstraction in newly developed code, the greater the number of reused modules and the greater the frequency of their reuse.[55] Other Japanese firms were also paying increasing attention to advanced programming concepts and systematically teaching these to employees.[56]

In addition, given the percentage of costs devoted to maintenance over the lifetime of a software product, a firm should clearly want to maximize its capabilities in this area. The factory environment should make it easier for a company to enforce the key strategies usually cited for improving software maintenance: clearly structured and modularized program designs and coding; documentation sufficient so that future programmers can understand the code, either to fix problems or to add functions; thorough testing in order to catch bugs early, before they become too expensive and perhaps impossible to fix; and use of already-generated (and tested) modules of code.[57]

Process Efficiency Through Teamwork and Communication

One of the key and frustrating conclusions in Brooks' testimony was that the usefulness of "man-months" as a unit of measurement for estimating time and manpower on software projects was a myth. His experience was that adding people to a big project increased the amount of time that had to be spent on training the new people and on communicating: "Men and months are interchangeable commodities only when a task can be partitioned among many workers with no communication among them...Adding more men then lengthens, not shortens, the schedule."[58] Brooks thus came to prefer only "a few good minds doing design and construction," rather than the 1000 or so people who worked on OS/360.[59]

But, if an integrated engineering and production environment -- a "factory" as defined in this paper -- truly facilitated standardization, division of labor, general use of good tools and practices, and effective communication and teamwork, then it might make Brooks' observation much less relevant, or at least increase the size of teams that can still be effective and expand management alternatives for dealing with tardiness or other problems. In fact, some managers of software factories in Japan appear to have already realized this. The two individuals responsible for developing Hitachi's software factories, when asked specifically what they do when projects are late, admitted they do not actually follow Brooks' advice. They **add** people -- not just anybody, it is true, but their best available people -- and this has proved to be more effective than not adding people,

at least within their factory system.[60] Additional questions asked in the survey, as well as public information, also indicate Japanese software managers can estimate time and manpower needs with remarkable accuracy, using historical databases relying on man/month measures.

Organizational Focus on Productivity and Quality

One might argue that, when large numbers of people are involved, a factory approach is the **best** way to bring together the necessary discipline and integration of tools, procedures, and management controls to facilitate standardization and other measures that allow managers and engineers to increase productivity and the quality (lack of defects) of their products. On the one hand, in facilities where managers have precise historical data on the capabilities and tools their people possess, scheduling and cost estimates, as well as personnel assignments should be more effective and easier to do. In the case of Toshiba, an entire factory system has been specifically maximized to deliver software on time and with minimal bugs, as well as develop and reuse enormous amounts of code in customized products, leading to lines-of-code productivity levels that far exceeded figures reported for other firms.[61] As in hardware factories, the productivity advantages of reusing standardized components, or even procedures and tools, often go far beyond simple output per worker on a given day, by saving engineering and manufacturing man-hours in design, inspection, rework needed to correct mistakes in the design or production of new components, and long-term maintenance.[62]

The type of data base provided for in the factory environment on numbers and types of bugs, as well as solutions, also provides a powerful means of learning how to produce higher quality products within certain cost targets. This type of defect and solution analysis, compbined with productivity and cost estimation, has proved useful in numerous "hard" industries such as automobiles. Product development teams often do the same; and there is no reason why engeering organizations cannot continue to improve their capabilities inthis area. In fact, evidence from the survey and other articles from Japan indicate that it is possible to make significant improvements in software quality control and that, as in other industries, improving quality improves productivity.[63]

Reducing Waste and Redundancy

A loosely managed engineering organization may indeed produce highly creative products. But, if there are insufficient directions, communications, and controls, different areas of the same organization might repeatedly design similar or substitutable tools, procedures, or even components--continually "reinvent the wheel" -- unnecessarily. A factory approach would reduce this type of wasteful, redundant behavior on the part of both managers and engineers. Companies need not leave the elimination of these dysfunctions to chance alone.

For example, if an individual project manager insists it is too expensive or time consuming for his or her group to write "inventory-quality software modules" for program libraries, the organization will create these reusable

assets only at the discretion of individuals, not as a matter of policy. Creating an environment where standardized components can be developed and reused seems to require that programmers design, code, and document modules in a standardized way suitable for reusability; place them in an easily accessible program library or central production data base; and then reuse them consistently when writing new programs. These three requirements are primarily matters of strategy and policy; and the third seems essential to justify the efforts expended and tradeoffs involved in standardization, documentation, and tool development.

Options for "Flexibility"

Nor do the technologies and techniques developed for the factory environments have to static. Software-development tools and systems at TRW, IBM, and SDC, as well as Hitachi and NEC have evolved incrementally and changed rather significantly over time. The factory approach also need not prevent an organization from pursuing, simultaneously or subsequently, other organizational alternatives as its needs change. For example, should a company want to experiment with different approaches to program design and management, including bringing some development activities closer to customers (decentralizing), it might set up separate departments or subsidiaries.

Fujitsu, Hitachi, and NEC alone in recent years have established nearly a hundred subsidiaries, to produce a variety of systems and applications software in less restricted and less centralized environments. These firms

also use R&D laboratories, such as NEC's Software Products Engineering Laboratory or Hitachi's Systems Development Laboratory, to experiment with different software technologies, separating these activities from "production" activites in the software factories. Hitachi has also added new departments for artificial intelligence and graphics to its systems software factory. In addition, Japanese computer manufacturers have actively transferred their factory-type programming systems and procedures to selected subsidiaries as well as other company divisions.[64]

CONCLUSIONS

1. THE SPECTRUM EXISTS: TECHNOLOGY & POLICY

Perhaps the most important finding of this study is that engineering organizations do follow different approaches to mixing their use of technology-based tools and strategy-based policies or methods in order to rationalize their development activites. One might think of the distinction between a factory and a job shop or laboratory as emerging from (1) a management strategy determined to take advantages of scale, scope, or other factors that might promote productivity and quality improvements; (2) a technological infrastructure facilitating this strategy; and (3) a management-policy and methodological infrastructure effectively integrating the technology with management objectives and people operating within or upon the system.

2. SOFTWARE IS NO EXCEPTION

The second major conclusion is that software-development organizations are not exceptions; they, too, exhibit different mixes of technology, policies, and thus overall strategies and implementation effectiveness. There are aspects of software development that resemble both engineering (design, maintenance) and manufacturing (coding, testing, assembly of exisitng modules with new code); the entire development cycle of software is also highly integrated and can be managed with varying degrees of discipline and standardization, tool-intensity, and the like.

Implicit or explicit proponents of the factory model appear to view this approach as a strategic way to accommodate shortages of programmers and the increasing demand for high-quality, often customized software. The rationale one might use to justify this attempt at process improvement in software engineering seems not so different from what many hardware-producing organizations learned decades ago: As companies accumulate knowledge and experience with product development and manufacturing, they should be able to raise the development process from activities performed by a few highly skilled individuals, operating in an environment analogous to a hardware job-shop, to activities performed systematically by individuals in an organization capable of historical learning, and perhaps some economies of scope or scale of activities. The latter organization might also benefit from the same advantages as any large-scale producer of products using carefully analyzed, controlled, and standardized processes and inputs, and try to avoid tradeoffs in product quality or features.

A recent article by an IBM manager on the evolution of software tools and techniques argues as well that firms are increasingly recognizing it is time to focus more on improving software process management. This asserts that software engineering began in the 1960s with a management focus on the individual engineer or programmer and a technological focus on techniques such as structured programming. In the next stage, roughly during the 1970s, experimental techniques became more formal methodologies, such as stepwise refinement or structured analysis, while managers turned more of their attention to understanding better the processes involved in each step of software life-cycle development (for example, from basic design

through maintenance).[65] But only in the most recent stage (the 1980s), in this interpretation, have companies pushed technology development more toward improving software environments, and have managers shifted their concerns to process-management issues.[66]

A technological and management change in focus -- from the individual and individual tools and techniques, to the organization and process management -- reflects a movement one might expect with any product and process as firms accumulate experience and as product technologies tend to stabilize, as suggested earlier in this paper for software. Managers then can become more "strategic" in two respects: recognizing the need to integrate better technology, policies, and organizational structure with the overall goals of the company or business unit; and perceiving these issues on levels that distinguish long-term from short-term tradeoffs, in the sense that each decision brings with it certain costs and benefits.[67] One of the objectives of managers should also be to minimize tradeoffs, such as product cost versus quality or functionality.

3. FLEXIBLE MANUFACTURING CONCEPTS & TOOLS IN ENGINEERING

As discussed earlier in this paper, it is a mistake to think of the factory model as requiring rigidity in processes or products, or technological features over time. Manufacturing and engineering organizations in "hard" industries have found numerous ways to increase their ability to rationalize development or production while enhancing their ability to introduce variety in final products. Toyota's small-lot production system exploits

standardization and discipline in components and procedures, as well as easily changeable equipment, to produce a variety of products each in small or large volumes with astounding levels of productivity. The histories of how manufacturing firms have improved productivity and product quality may themselves provide valuable lessons for engineering. In addition, group technology practices; computer-aided design, engineering, and manufacturing tools; as well as entire flexible manufacturing systems as in the machine-tool industry, all incorporate concepts and technologies that are already being applied in software but can be applied more widely, without necessarily sacrificing the need for products to meet a variety of customer needs. These themes, and the actual practices and technologies used in software facilities that resemble flexible factories, are being treated in the case studies accompanying this paper.

4. THE JAPANESE MAY BE LEADING IN SOFTWARE MANAGEMENT

The survey indicated that Japanese firms or individual facilities, on average, were considerably ahead of most U.S. firms in the application of factory-type policies to large-scale software development. The United States has seen its lead in product engineering and manufacturing skills evaporate in a broad range of industries: steel, shipbuilding, consumer electronics, automobiles, semiconductors, computer peripherals, and machine tools, to name some of the more obvious. The Japanese have also been making significant strides in computer hardware, including supercomputers.[68] Software development benefits from, in addition to a measure of "creativity," disciplined procedures, basic mathematical skills, systematic training, support

tools, and effective group communication as projects get larger. There is no reason why the Japanese cannot do well in software; the evidence provided here and in cases studies, moreover, suggests they may be leading in tool application and management control.

Arguments will continue whether software development as a process is more like art than science or engineering, or more like research and development than manufacturing in a hardware environment. There may be strong "cultural" or historical tendencies of U.S. software engineers, especially those who are largely self-educated, to view software as largely a form of art. Japanese managers and programmers may not have acquired similar tendencies; and the historical commitment established in other fields of firms such as NEC, Toshiba, Hitachi, and Fujitsu to promote their skills in engineering, manufacturing, and quality control may in fact be overcoming any predisposition of their software people against a factory approach. These issues will be treated in studies of individual firms.

But the bias of managers regarding this question is not trivial. It may be at least as important as the characteristics of the technology, to the extent that managerial (and worker) attitudes constrain or facilitate the ability of organizations to improve the performance of their people and systems, as well as the features of their products. But even if one assumes aspects of the software-development process, such as detailed design in a large program, are essentially similar to a creative product development, a firm can still apply strategic measures to technology development and utilization, and to policies or procedures, to "rationalize" and improve overall

performance. Moreover, if some firms deemphasize discipline and cooperation while focusing essentially on the individual engineer, the individual tool, or the final product, then they may not be fully developing -- that is, compared to some of their competitors -- **organizational capabilities** to maximize the performance of their technical people and invested resources.

The apparent superceding of Brooks' "law" in a Japanese software factory encourages the belief that it is possible to manage software and perhaps engineering activities more strategically, through a better integration of technology and management policies. That some firms are closer than others to implementing flexible-factory models also suggests there is nothing inherent in the technology to prevent the introduction of this approach. If managers claim a factory environment as proposed in this paper is undesirable because of the tradeoffs it may entail, they make a strategic judgement. If they insist it is technologically too difficult, they exhibit a conceptual or even emotional bias that, in view of the present study, merits serious reconsideration.

APPENDICES

Formal Hypothesis Tests (1-3)

Hypothesis 1: Accept (Facilities' scores follow a normal distribution)

Variable:	Totals	Technology	Policy
Sample Size:	38	38	38
Average:	66.89	71.18	64.45
Median	70	77	67
Mode	86	84	43
Standard Deviation:	17.26	17.40	18.59
Range	76	70	82
Kurtosis:	0.31	0.03	0.02
Standardized Kurtosis:	0.40	0.03	0.03

Hypothesis 2: Accept (There is no significant difference in the technology scores of the U.S. and Japanese facilities.)

Two Sample Analysis Results: Technology Scores

	Japanese	U.S.	Pooled
Observations:	17	21	38
Average:	75.47	67.71	71.18
Std. Deviation:	15.10	18.68	17.18
Median	78	72	77

Null Hypothesis (Ho): There is no significant difference in the technology scores of the U.S. and Japanese facilities.

Confidence Interval for Difference in Means: 95% or 99%
Hypothesis Test for Ho: Diff = 0 Computed t statistic = 1.38347
vs. Alt: Not Equal Sig. Level = 0.175039
at Alpha = 0.05 or 0.01 Do Not Reject Null Hypothesis

Hypothesis 3: Accept (Japanese score for policy/methodology infrastructure as well as Japanese total score significantly higher than the comparable U.S. scores)

Two Sample Analysis Results: Policy Scores

	Japanese	U.S.	Pooled
Observations:	17	21	38
Average:	74.88	56.0	64.45
Std. Deviation:	15.65	16.61	16.19
Median	73	59	67

Null Hypothesis (Ho): Japanese score for policy/methodology infrastructure as well as Japanese total score are not significantly higher than the comparable U.S. scores.

Confidence Interval for Difference in Means: 95% or 99%
Hypothesis Test for Ho: Diff = 0 Computed t statistic = 3.5745
vs. Alt: Not Equal Sig. Level = 1.02177E-3
at Alpha = 0.05 or 0.01 Reject Hypothesis

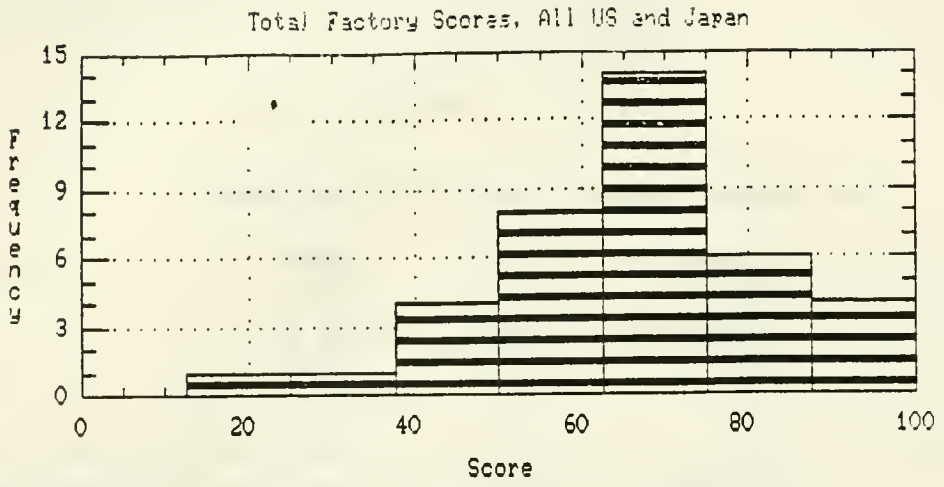
Two Sample Analysis Results: Total Scores

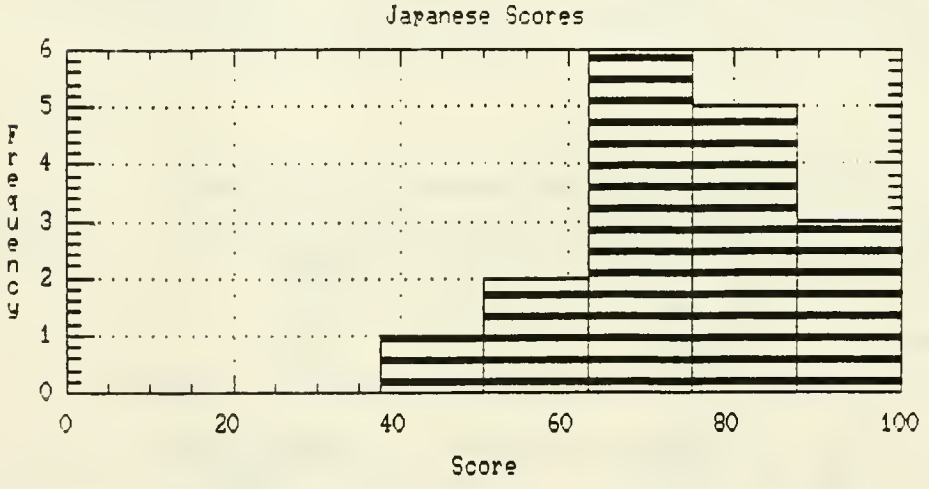
	Japanese	U.S.	Pooled
Observations:	17	21	38
Average:	75.18	60.0	66.79
Std. Deviation:	15.26	16.03	15.70
Median	75	62	70

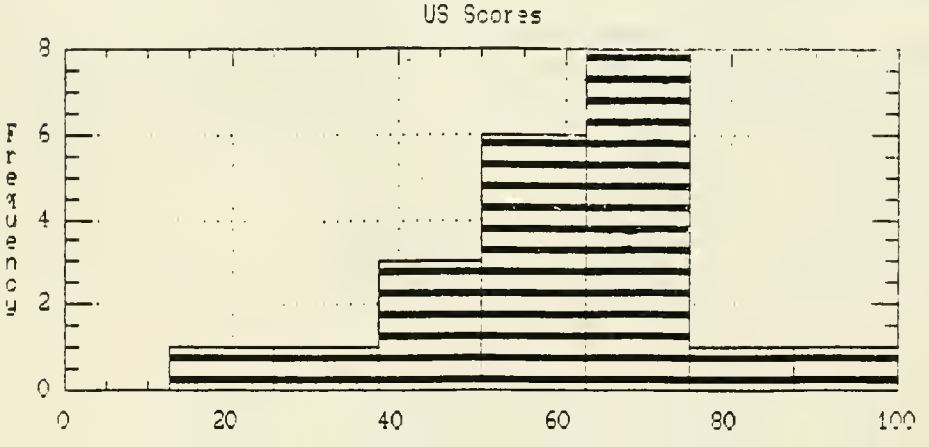
Confidence Interval for Difference in Means: 95% or 99%
Hypothesis Test for Ho: Diff = 0 Computed t statistic = 2.96441
vs. Alt: Not Equal Sig. Level = 5.35201E-3
at Alpha = 0.05 or 0.01 Reject Hypothesis

Sample Characteristics:

Variable:	Japanese	U.S.
Sample Size:	17	21
Average:	75.47	60
Median	78	62
Mode	84	61
Standard Deviation:	15.10	16.03
Range	57	65
Kurtosis:	0.35	0.80
Standardized Kurtosis:	0.29	0.75


Total Factory Scores, All US and Japan
Frequency
15
12
9
6
3
0
0
20
40
60
80
100
Score


Japanese Scores
Frequency
6
5
4
3
2
1
0
0
20
40
60
80
100
Score


US Scores
Frequency
8
6
4
2
0
0
20
40
60
80
100

MEANS AND STANDARD DEVIATIONS FOR SURVEY QUESTIONS

SURVEY ANSWERS KEY:

4 = CAPABILITY OR POLICY IS FULLY USED OR ENFORCED
3 = CAPABILITY OR POLICY IS FREQUENTLY USED OR ENFORCED
2 = CAPABILITY OR POLICY IS SOMETIMES USED OR ENFORCED
1 = CAPABILITY OR POLICY IS SELDOM USED OR ENFORCED
0 = CAPABILITY OR POLICY IS NOT USED

n = 38 (Jap. = 17, U.S. = 21)

I. TECHNOLOGY/FACILITY INFRASTRUCTURE

ALL COMPANIES/FACILITIES

Question	All Companies		Japanese		U.S.	
	Mean	S. Dev.	Mean	S. Dev.	Mean	S. Dev.
A	3.47	0.62	3.38	0.65	3.55	0.58
B	3.45	0.71	3.69	0.48	3.25	0.79
C	3.07	1.02	2.97	0.87	3.15	1.13
D	2.55	1.04	2.99	0.67	2.20	1.14
E	2.68	1.18	2.44	1.17	2.88	1.15
F	3.04	1.08	3.40	0.86	2.75	1.14
G	2.67	1.25	2.94	1.08	2.45	1.34
H	1.85	1.06	2.37	0.82	1.43	1.04

APPLICATIONS COMPANIES/FACILITIES

A	3.35	0.62	3.15	0.71	3.50	0.50
B	3.31	0.76	3.58	0.52	3.12	0.84
C	2.97	1.10	2.70	0.93	3.16	1.18
D	2.54	0.93	2.83	0.67	2.33	1.03
E	2.65	1.25	2.20	1.27	2.96	1.14
F	2.84	1.18	3.23	0.98	2.56	1.24
G	2.54	1.36	2.75	1.66	2.39	1.39
H	1.86	1.04	2.38	0.64	1.50	1.12

SYSTEMS COMPANIES/FACILITIES

A	3.68	0.55	3.71	0.36	3.64	0.69
B	3.68	0.52	3.86	0.35	3.50	0.60
C	3.25	0.84	3.36	0.58	3.14	1.03
D	2.57	1.19	3.21	0.59	1.93	1.29
E	2.75	1.05	2.79	0.92	2.71	1.16
F	3.39	0.74	3.64	0.58	3.14	0.79
G	2.89	1.00	3.21	0.59	2.57	1.21
H	1.82	1.08	2.36	1.03	1.29	0.84

II. METHODOLOGY & POLICY INFRASTRUCTURE

ALL COMPANIES/FACILITIES

Question	All Companies Mean	All Companies S. Dev.	Japanese Mean	Japanese S. Dev.	U.S. Mean	U.S. S. Dev.
A	1.77	1.20	1.85	1.14	1.71	1.24
B	2.50	1.30	2.98	1.08	2.16	1.34
C	3.33	0.89	3.55	0.65	3.18	0.99
D	2.00	1.11	2.05	1.21	1.96	1.03
E	2.81	1.07	3.20	0.95	2.54	1.06
F	2.67	0.98	3.30	0.60	2.21	0.94
G	2.46	1.16	2.75	1.17	2.25	1.11
H	2.07	1.28	2.88	0.85	1.50	1.22
I	1.99	1.04	2.58	0.87	1.57	0.94
J	2.53	1.25	2.90	1.24	2.26	1.19
K	1.94	1.27	2.33	1.43	1.67	1.05
L	2.90	1.17	2.75	1.29	3.01	1.06
M	3.18	1.10	3.65	0.63	2.85	1.24
N	2.71	1.03	3.13	0.75	2.42	1.10
O	2.61	1.13	3.48	0.53	2.00	1.04

APPLICATIONS COMPANIES/FACILITIES

A	1.77	1.20	1.85	1.14	1.71	1.24
B	2.50	1.30	2.98	1.08	2.16	1.34
C	3.33	0.89	3.55	0.65	3.18	0.99
D	2.00	1.11	2.05	1.21	1.96	1.03
E	2.81	1.07	3.20	0.95	2.54	1.06
F	2.67	0.98	3.30	0.60	2.21	0.94
G	2.46	1.16	2.75	1.17	2.25	1.11
H	2.07	1.28	2.88	0.85	1.50	1.22
I	1.99	1.04	2.58	0.87	1.57	0.94
J	2.53	1.25	2.90	1.24	2.26	1.19
K	1.94	1.27	2.33	1.43	1.67	1.05
L	2.90	1.17	2.75	1.29	3.01	1.06
M	3.18	1.10	3.65	0.63	2.85	1.24
N	2.71	1.03	3.13	0.75	2.42	1.10
O	2.61	1.13	3.48	0.53	2.00	1.04

SYSTEMS COMPANIES/FACILITIES

A	2.21	1.03	2.43	1.08	2.00	0.93
B	2.50	1.04	2.86	0.95	2.14	0.99
C	3.86	0.40	3.93	0.17	3.79	0.52
D	2.43	1.13	2.79	1.19	2.07	0.94
E	3.21	0.65	3.36	0.69	3.07	0.56
F	2.75	0.96	3.29	0.52	2.21	0.99

G	2.54	1.13	2.71	0.84	2.36	1.33
H	2.21	1.47	3.57	0.42	0.86	0.69
I	2.21	1.24	2.64	1.19	1.79	1.13
J	2.54	1.20	3.36	1.03	1.71	0.70
K	2.18	1.42	3.14	1.16	1.21	0.92
L	3.36	0.61	3.36	0.79	3.36	0.35
M	3.61	0.43	3.71	0.36	3.50	0.46
N	2.54	1.19	3.07	0.94	2.00	1.16
O	2.39	1.18	2.86	1.38	1.93	0.68

Question: **How do you measure the "performance" of your project managers?**

Japanese Applications:

1: Quality, cost, delivery; leadership, presentation, negotiation ability
2: Cost: Cost/person and profit/person
3: Productivity and management of personnel
4: Productivity: released loc/man month; Cost Productivity: cost/released lines of code

US Applications:

1: Don't
2: Quality of product (error-free, minimal rework); delivered on time; within budget; satisfaction of customer
3: Customer satisfaction in meeting schedules, staying within budget, and meeting performance requirements
4: Cost/schedule
5: Meeting schedule and user requirements; cost is secondary
6: 1) When they indicate coding is completed; 2) number of bugs reported in-house or beta-site after coding is completed
7: Budget dollars, schedule performance, technical scope, group turnover, customer satisfaction
8: Budget and schedule plus quality
9: Cost and scedule performance indexes; rate charts; milestone completions; lead division interface; customer satisfaction; personnel management activities
10: Subjectively
11: Schedule aherence
12: Schedule, cost, number of modules forecast vs. actual (yet to be implemented measurements should include errors or quality of code, and ease of maintenance)
13: Adherence to schedule; molding of a cohesive group or not

Japan Systems:

1: Growth rate of members through the project; productivity of the project; quality of the project; experience
2: 170 items related to project control (quality, delivery, cost).
3: Code trace and stress testing with tools or terminals

US Systems:

1: Meeting functional requirements; schedule
2: Revenue/cost
3: Cost/schedule performance
4: Achieving schedule completion and the amount of defects discovered in the testing process

Question: How do you measure the "performance" of your programmers?

Japanese Applications:

1: Productivity and product quality; source program update frequency of programmers
2: Capability: loc, specification pages, and test items per person/month, with adjustments for type of product, correctness, bugs, etc.; separate categories for analysts, designers, programmers, and test engineers
3: Steps/time
4: Released loc/hour or loc/month
5: Acurracy and quality of the product

US Applications:

1: Schedule; LOC/Complexity; Errors
2: Amount of work completed; correctness of work done; ability to meet schedule; innovation in solving technical problems
3: Comparisons to peers based on use of performance appraisal forms (20 attributes); feedback from project supervisors concerning performance; letters of recognition
4: Cost, schedule, quality
5: Project leaders write "Performance Aanalysis" reports on each member at the end of the project
6: As a function of quality, productivity, contribution
7: Perception. Measure against goals after the period of measurement is completed.
8: Through A/O process; speed, accuracy, and quality of final program product. Additionally, teamwork and schedule, and finally, documentation.
9: Performance to schedule. Quality of code.
10: Productivity (sloc/mo), meeting schedules, growth
11: Subjectively
12: Judgement based on software size, complexity, implementation environment, quality, completion relative to projected cost and schedule, hours worked, cooperativeness, internal and external communication skills.
13: Schedules, errors in completed code, documentation, maintainability of code, installability, design
14: Ability to design, function in a group atmosphere, code from specs, adherence to schedules

Japanese Systems:

1: Productivity & quality of programs written; experience
2: Discovered bugs/1000 loc
3: Lines of code/month

US Systems:

1: Closeness to functional and schedule targets, with acceptable performance
2: No formulas, but a look a productivity, quality, and leadership
3: Schedule, product quality
4: Schedule achievement and technical review of design, code, test output; measured number of defects

REFERENCES

1. I am indebted to numerous individuals for suggestions that I have incorporated into this paper. In the U.S.: John Pilat of Data General; Wendy MacKay of DEC; Fred George and Mark Harris of IBM; Joel Moses, Chuck Sabel, Ed Roberts, Eric von Hippel, Tom Allen, and David Finnell of MIT; Donald McNamara of GE; Suzanne de Treville of Harvard, MIT and the Helsinki School of Economics. In Japan: Shibata Kanji of Hitachi; Azuma Motoei of NEC; Yoshida Tadashi of Fujitsu; Matumoto Yoshihiro of Toshiba. I would also like to thank Peter Freeman and Veikko Seppannen of the University of California, Irvine, for several important comments and literature references.

2. See Barry Boehm, "Software Engineering (1976)" in Yourdon, p. 328.

3. See Alfred D. Chandler, Jr., **The Visible Hand: The Managerial Revolution in American Business** (Cambridge, MA, Harvard University Press, 1977), pp. 50-80.

4. See William J. Abernathy and James Utterback, "Dynamic Model of Process and Product Innovation," **Omega**, Vol. 3, No. 6, 1975, pp. 639-657; William J. Abernathy and Kenneth Wayne, "Limits of the Learning Curve," **Harvard Business Review**, September-October 1974, pp. 109-119; and Robert H. Hayes and Steven C. Wheelright, "Link Manufacturing Process and Product Life Cycles," **Harvard Business Review**, January-February 1979, pp. 133-140, and **Restoring Our Competitive Edge: Competing through Manufacturing** (New York, John Wiley & Sons, 1984), pp. 197-298.

5. An historical analysis of the evolution of the Toyota production system, and productivity comparisons for the major Japanese and U.S. automakers, can be found in Michael A. Cusumano, **The Japanese Automobile Industry: Technology and Management at Nissan and Toyota** (Cambridge, MA, Harvard University Press, 1985), pp. 186-319.

6. See Michael E. Porter, **Competive Strategy: Techniques for Analyzing Industries and Competitors** (New York, The Free Press, 1980), pp. 34-46, and **Competitive Advantage: Creating and Sustaining Superior Performance** (New York, The Free Press, 1985), p. 18.

7. See Michael J. Piore and Charles F. Sabel, **The Second Industrial Divide: Possibilities for Prosperity** (New York, Basic Books, 1984). For more detailed discussions of flexible manufacturing systems see Paul Kinnucan, "Flexible Systems Invade the Factory," **High Technology**, July 1983, pp. 32-43; National Research Council, **The U.S. Machine Tool Industry and the Defense Industrial Base** (Washington, D.C., National Academy Press, 1983); Ramchandran Jaikumar, "Flexible Manufacturing Systems: A Managerial Perspective," Harvard Business School Working Paper #1-784-078,

January 1984, and "Postindustrial Manufacturing," **Harvard Business Review**, November-December 1986, pp. 301-308. A general review of these new technologies is J. Meredith, "The Strategic Advantages of New Manufacturing Technologies for Small Firms," **Strategic Management Journal**, Vol. 8, No. 3, May-June 1987, pp. 249-258.

8. In a list of 31 FMS installations in the U.S., two-thirds were in these 5 fields (motors -- 6, aircraft -- 6, machine tools -- 2, construction and agricultural equipment -- 3, defense systems -- 3). See Diane Palframan, "FMS: Too Much, Too Soon," **Manufacturing Engineering**, March 1987, p.36.

9. See Harvard Business School, "VLSI Technology, Inc. (A)" (Case Study 0-686-128, 1986).

10. See Nancy L. Hyer and Urban Wemmerloc, "Group Technology and Productivity," **Harvard Business Review**, July-August 1984, pp. 140-149; and Harvard Business School, "Note on Group Technology and Cellular Manufacturing" (Boston, Harvard Case #9-686-098, 1986).

11. These estimates from M. V. Zelkowitz et al., **Principles of Software Engineering and Design** (Englewood Cliffs, N.J.: Prentice-Hall, 1979), p. 9. Cited also in Frank, p. 22; Ramamoorthy, p. 193.

12. See A. Zavala, "Research on Factors that Influence the Productivity of Software Development Workers," SRI International, June 1985.

13. Ibid., pp. 10-11.

14. Data from Dataquest and **Businessweek**, quoted in **Businessweek**, 11 May 1987, p. 149.

15. U.S. Department of Commerce, **A Competitive Assessment of the U.S. Software Industry** (U.S. Dept. of Commerce, International Trade Administration, Washington, D.C., 1984), pp. 19-24, 34-35.

16. See "The Free-For-All Has Begun: Software Companies Large and Small are Invading One Another's Turf," **Businessweek**, 11 May 1987, pp. 148-159.

17. See H. Aiso (Keio University), "Overview of Japanese National Projects in Information Technology," International Symposium on Computer Architecture, Lecture 1, 2 June 1986, Tokyo. Data from a 1986 Japanese-language white paper published by the Japan Information Service Industry Association.

18. **Competitive Status of the U.S. Software Industry**, pp. 34-35, 40, 42.

19. A classic article on this subject is B.W. Boehm, "Software Engineering," **IEEE Transactions on Computers**, Vol. C-25, No. 12, December 1976, pp. 1226-1241.

20. One basic textbook, in a section titled, "Is the Programmer a Scientist, an Engineer, or an Artist," sees the programmer as primarily an "engineer," preferring the scientist label for those doing research or creating new technology, and the "artist" label for "a small number of people who excel at their craft to an extraordinary degree," such as "Michelangelo." See Martin Shooman, **Software Engineering: Design, Reliability, and Management** (New York: McGraw-Hill, 1983), pp. 5-8.

21. See C.V. Ramamoorthy et al., "Software Engineering: Problems and Perspectives," **Computer**, October 1984, p. 205.

22. This argument is developed in R. Goldberg, "Software Engineering: An Emerging Discipline," **IBM Systems Journal**, Vol. 25, Nos. 3/4, 1986, pp. 334-353.

23. See the earlier works by Abernathy, Utterback, Wayne, Hayes, and Wheelright.

24. U.S. Department of Commerce, **A Competitive Assessment of the U.S. Software Industry** (Washington, D.C., International Trade Administration, 1984), p. 34.

25. Data from TRW in the U.S. and Toshiba and Hitachi in Japan indicate that as much as 60% or more of applications programs and 90% of new releases in systems software appear to be redundant across different products, providing potentially high volumes. For a discussion of the TRW study, see Werner L. Frank, **Critical Issues in Software** (New York: John Wiley & Sons, 1983), pp. 74-75. The original citation is Robert Lanergan and Denis Dugan, "Requirements for Implementing a Successful Reusable Code Productivity System," Raytheon Co. Missile Systems Division, 1980. Data for Toshiba can be found in Kim, p. 33, and Matsumoto (1986), p. 5. The 90% figure is for Hitachi Software Works, Shibata interview. Company names in the survey cannot be released do to confidentiality agreements.

26. Some companies, such as Hitachi, formally organize product-engineering and manufacturing departments within the same physical facility. For example, for computer hardware see Hitachi Seisakusho Kabushiki Kaisha (Hitachi Ltd.), **Kanagawa kojo 15 nen no ayumi** (15-year history of the Kanagawa Works, 1978), pp. 129-132; for semiconductors, **Musashi kojo 20 nen no ayumi** (20-year history of the Musashi Works, 1978), Appendix.

27. See Alfred D. Chandler, Jr., **The Visible Hand: The Managerial Revolution in American Business** (Cambridge, M.A.: Harvard University Press, 1977), pp. 50-80.

28. See Harvey Bratman and Terry Court (System Development Corporation), "The Software Factory," Computer, May 1975, pp. 28-29. A more detailed discussion of this can be found in H. Bratman and T. Court, "Elements of the Software Factory: Standards, Procedures, and Tools," in Infotech International Ltd., **Software Engineering Techniques** (Berkshire, England: Infotech International Ltd., 1977), pp. 117-143.

29. Intervews with David Deaver, SDC Manager, and Clarence Starkey, SDC Manager, 10/3/86. Also, interview with and SDC/Unisys director of software engineering, 4/1/87. Name withheld by request of the interview subject. My thanks to David Finnell for conducting this interview under my direction, as part of a thesis project. For an analysis of SDC, see Michael Cusumano and David Finnel, **A U.S. "Software Factory" Experiment: System Devleopment Corporation** (MIT Sloan School of Management Working Paper, 1987).

30. The NATO conference reference comes from Donald McNamara, Program Manager, Corporate Information Technology, General Electric Company, "Software Factories," Lecture given at the Wang Institute of Graduate Studies, Lowell, MA., 2 February 1987. See later references for Toshiba and Hitachi.

31. In developing their production-control system during the late 1960s and 1970s, Hitachi managers also relied on reports published on SDC's approaches to standardizing software development (interview with Shibata Kanji, Manager, Engineering Department, Hitachi Software Works, 9/19/85). A major article written in 1981 by the manager primarily responsible for developing the Toshiba facility also cited the SDC factory experiment as a precedent; see Yoshihiro Matsumoto (Toshiba), "Management of Industrial Software Production," **Computer**, February 1984, p. 318 fn 2. Along with developing their own technology, NEC engineers extensively studied American software techniques, including SDC's estimating model and then the Software Factory during the 1970s as approaches to cost-estimation and project control (interview with NEC Vice-President Mizuno Yukio, 9/26/85). See also Iwamoto Kanji and Okada Masashi (NEC), "Purojekuto kanri no tsuru" (Project control tools), **Joho shori** (Information processing), Vo. 20, No. 8, p. 721. Iwamoto was a member of the Computer Systems Research Laboratory, part of the Central Research Laboratories; he joined the Software Product Engineering Laboratory when NEC created this in 1980. Okada was then in the systems engineering department of NEC Software Ltd., a subsidiary.

32. See Marvin Zelkowitz et al., "Software Engineering Practices in the US and Japan," **Computer**, June 1984. The authors, who studied about 30 organizations, concluded that, "we found the level of technology used by the Japanese to be similar to US practices, but with some important differences...we found that Japanese companies typically optimize resources across the company rather than within a single project... Thus, tool development and use is more widespread in Japan" (p. 63).

33. Examples of recent reports on Japanese activities in software include the following, in addition to "Software in Japan," **Electronic Engineering Times**, 11 February 1985 (quoted in the body of this article):

A Competitive Assessment of the U.S. Software Industry (U.S. Dept. of Commerce, International Trade Administration, Washington, D.C., 1984), p. vi:

> "... more widespread use of software engineering techniques by the Japanese may enable them to have an edge in producing lower cost, higher quality (i.e. error free) software, faster than their U.S. counterparts.

Robert Haavind, "Tools for Compatibility, " **High Technology**, August 1986, pp. 34-42:

> "A major national program and many individual company projects aim to improve software quality and productivity by developing techniques that range from reusing parts of previous programs to fully automating software production.
> ... Discipline and adherence to rigid software engineering practices are common in Japan, making the climate for automated programming more favorable than it might be in a more freewheeling setting. Software is often developed in regimented, factorylike settings different from anything in the United States.
>
> ... Although Japanese companies are making important progress in boosting programmer productivity while turning out essentially bug-free software, NEC...is puzzled that [there is] so little similar activity in the United States. The U.S. may retain some mystique from the early days of computers when software grew up as a black art, whereas Japan, which only recently recognized that software is the key to expanding the future uses of computers, is pushing this technology after great strides have been made in software engineering.

"Japan's Push to Write 'World-Class' Software," **Businessweek**, 2 7 February 1984, pp. 96-98:

> "...[I]t would be dangerous to write off the Japanese as competitors in world-class software... 'People are misreading the capability of the Japanese when they say Japanese can't build good software,' maintains Joseph C. Berston, president of Comstute Inc., a software consulting firm based in Japan. Indeed, Japanese companies may actually have some advantages over their U.S. rivals. Because of the legendary thoroughness of Japanese workers, 'the finished product here is better, more reliable, and easier to maintain,' says consultant Berston. Labor costs are also lower for Japanese software makers... In addition to that salary differential, the Japanese claim their programmers are 10% to 15% more productive than their U.S. counterparts because of Japanese investments in program-development aids. To widen that margin, they are now building software factories that give their programmers access to even

more sophisticated tools."

Bro Uttal, "Japan's Persistent Software Gap," **Fortune**, 15 October 1984, pp. 151-160:

"NEC and Hitachi in particular have developed arsenals of programs to automate programming. They're also cutting costs and improving quality with management techniques for catching software defects early, before the bugs burrow deep into finished programs... The mainframers are relying heavily on software 'factories' to cut costs and improve quality. The idea is to apply mass-production techniques to writing programs that are mostly custom-tailored. Hitachi built the first factory, a five-story structure housing 3,000 programmers, in 1969. Fujitsu and NEC followed suit with smaller centers for 1,000 to 2,000 workers. Together the Big Three now boast ten factories. Such factories, rare in the U.S., treat software as an industrial product, not as a form of art... Though the final product may not be innovative, it's often highly reliable."

34. A recent paperback by a popular Japanese journalist was devoted solely to the appearance of software factories in Japan. See Shimoda Hirotsugu, **Sofutouea kojo** (Software factories) (Tokyo: Toyo Keizai Shimposha, 1986).

35. **A Competitive Assessment of the U.S. Software Industry** (U.S. Dept. of Commerce, International Trade Administration, Washington, D.C., 1984), p. 61.

36. "Software in Japan," **Electronic Engineering Times**, 11 February 1985, p. 1.

37. Kiriu Hiroshi, **Sofutouea sangyo no jitsuzo** (The actual status of the software industry), Tokyo, Nikkan Shobo, 1986, pp. 184-201.

38. There were as follows: Nippon Business Consultant (2,450 employees); Computer Service (3,745); Nihon Information Service (1,210); Hitachi Software Engineering (2,400); Toyo Information Systems (1,100); Intek (1,549); Fujitsu Aibi (1,256); Nihon Denshi Keisan (1,117); NEC Software (1,654); Ainesu (1,036); Hitachi Microcomputer Engineering (1,400); Nihon System Development (1,300); Nihon Systemware (1,100); Enjaeke (1,300); Nihon Computer Service Center (2,043); Data Process Consultant (1,000); Maruei Keisan Center (1,100). Source: Kiriu, pp. 50-58.

39. Based on an analysis of Kiriu; Takahashi Kenkichi et al., **Konkyuta gyokai** (The computer industry), Tokyo, Kyoikusha, 1985, pp. 161-210; Shimoda Hirotsugu, **Sofutouea kojo** (Software factories) (Tokyo: Toyo Keizai Shimposha, 1986); and Toyo Keizai Shimpo, **Kaisha shikiho** (Company quarterly reports), 1986.

40. According to **Datamation** (1 June 1985, pp. 58-120), in fiscal 1984, among Japanese computer manufacturers, NEC ranked first in software revenues ($299.9 million), followed by Fujitsu ($200) and Hitachi ($100). See also **A Competitive Assessment of the U.S. Software Industry**, p. 40.

41. Kiriu, pp. 78-91.

42. Kiriu, p. 50.

43. See Michael A. Cusumano, "Diversity and Innovation in Japanese Technology Management," in Richard S. Rosenbloom, ed., **Research on Technological Innovation, Management, and Policy** (Greenwich, Conn., JAI Press, Vol. 3, 1986), pp. 137-167; Kenichi Imai et al., "Managing the New Product Development Process: How Japanese Companies Learn and Unlearn," and "Commentary," in Kim B. Clark et al. eds., **The Uneasy Alliance: Managing the Productivity-Technology Dilemma** (Boston: Harvard Business School Press, 1985), pp. 330-381; Toshiro Hirota, "Technology Development of American and Japanese Companies," **Kansai University Review of Economics and Business**, March 1986, pp. 43-72.

44. Interviews with Mizuno Yukio, Vice-President, NEC, 9/26/85; Mr. Azuma Motoei, Manager, Software Management Engineering Dept., Software Product Engineering Laboratory, NEC, 9/26/85 and 7/28-29/86; and Yamaji Katsuro, Deputy General Manager, Software Division, Computer Systems Group, Fujitsu, 7/31/86. See also the discussion by Fujitsu Managing Director Mitsugi Mamoru, quoted in Shimoda, p. 82.

45. Regarding NEC, in English, see, for example, Yukio Mizuno (NEC), "Software Quality Improvement," **Computer**, March 1983, pp. 66-72, and "A Quantitative Approach to Software Quality and Productivity Improvement," (NEC Corporation, undated); and Tadashi Yoshida (Fujitsu), "Attaining Higher Quality in Software Development: Evaluation in Practice," **Fujitsu Scientific and Technical Journal**, Vol. 21, No. 3, July 1985, pp. 305-316. My discussion here is also based on several interviews with Mr. Yoshida. Each company also has extensive regulations and education programs for software quality. At Fujitsu, for example, these are described in "Sofutouea kaihatsu: hinshitu, seisansei kojo ni tusite" (Software development: concerning quality and productivity improvement), Fujitsu, Information Processing Group, No. 1 Software Division; and "Sofutouea no hinshitsu kanri" (Software quality control), Fujitsu, Computer Systems Group, Software Division, 11 September 1985.

46. IBM, for example, has its Santa Teresa Laboratory. See G.H. McCue, "IBM'S Santa Teresa Laboratory: Architectural Design for Program Development," **IBM Systems Journal**, Vol. 17, No. 1, 1978. Other systems software facilities in IBM, such as at Endicott and Poughkeepsie, New York, are also referred to as laboratories. DEC tends to use no labels, and Data General simply "facility."

47. An excellent summary of the state of the field is Goldberg's article in **IBM Systems Journal**. More specific articles on IBM practices can be found in issues from this journal in 1978, 1980, 1985, 1986. Another good summary of articles is H. Hunke, ed., **Software Engineering Environments**, (Amsterdam, North-Holland, 1981. For a specific articles on TRW, see Barry W. Boehm et al., "A Software Development Environment for Improving Productivity," **Computer**, June 1984, pp. 30-44; M.H. Penendo and A.B. Pyster, "Software Engineering Standards for TRW's Software Productivity Project," **Proc. Second Software Engineering Standards Application Wokshop**, May 17-19, 1983. Other papers and articles published in 1986-1987 in various **IEEE** and **ACM** tutorials and other forums are also dealing extensively with developments in software engineering environments at U.S. and Japanese firms, and issues raised in this paper regarding the benefits or disadvantages posed by a factory model.

48. Actual company names and numbers cannot be given out, due to confidentiality agreements. -- indicates either the information is not tracked or was not provided.)

49. These and other benefits of the factory approach, such as division of labor and standardization of procedures, should result in higher nominal productivity rates, such as lines of code in a given time period, although measuring this across different products written in different languages for different machines is problematic and has not been attempted in the survey.

50. My thanks to John Pilat of Data General for this observation.

51. See Yoshihiro Matsumoto (Toshiba), "A Software Factory: An Overall Approach to Software Production" (2 December 1986), forthcoming in Peter Freeman, ed., **Software Reusability** (IEEE Tutorial, 1987), p. 17 (unpublished draft).

52. See, for example, a recent **IEEE Tutorial** on **Software Reusability**, edited by Peter Freeman. **IEEE Transactions on Software Engineering** (SE-10, No. 5, 1984) was also a special issue on reusability and contains several useful article on U.S. and Japanese cases.

53. See Matsumoto's 1987 paper on Toshiba, as well as Cusumano's paper on Hitachi.

54. Incorporating high levels of abstraction in procedures and data types facilitates testing, debugging, and product enhancements, as well as maximizes the generality or reusability of modules. See Barbara Liskov and John Guttag, **Abstraction and Specification in Program Development** (Cambridge, MA: MIT Press, 1986), especially pp. 6-10, 40-42, 56-57, 308. Layering is somewhat of an alternative to strict top-down, structured design, which for over a decade has been the preferred approach since it tends to result in programs easier to modularize, verify, document, and maintain. I especially thank Joel Moses and Wendy Mackay for their comments on this. For discussions of top-down design and structured programming, see Dijkstra (1965); Bohm and Jacopini (1966); Baker (1972); Stevens, Myers, and

Constantine (1974); and Knuth (1977) collected in Edward Nash Yourdon, ed., **Classics in Software Engineering** (New York, Yourdon Press, 1979). See also Brooks, pp. 142-144.

55. Matsumoto (1984), pp. 61, 68.

56. Several papers from researchers and managers at NEC, mainly from the Software Product Engineering Laboratory, provide examples of this: Kanji Iwamoto et al., "Early Experiences Regarding SDMS Introduction into Software Production Sites," **NEC Research & Development**, No. 68, January 1983, especially pp. 51-60; Kanji Iwamoto and Osamu Shigo, "Unifying Data Flow and Control Flow Based Modularization Techniques," **Proceedings of the 23rd IEEE Computer Society International Conference (Compcon '81)**, September 15-17, 1981, pp. 271-277; Tateyuki Tsurutani, Osamu Shigo, and Touru Maejima, "SPOT: A Structured System Development System," **NEC Research & Development**, No. 40, January 1976, pp. 63-71; Osamu Shigo, Kanji Iwamoto, and Shinya Fujibayashi, "A Software Design System Based on a Unified Design Methodology," **Journal of Information Processing**, Vo, 3, No. 3, September 1980, pp. 186-196. For discussions of Hitachi, see Y. Futamura et al. (Central Research Laboratory, Hitachi), "Development of Computer Programs by Problem Analysis Diagram (PAD)," **Proceedings, Fifth International Conference on Software Engineering** (IEEE, 1981), pp. 325-332; M. Kobayashi (Systems Development Laboratory, Hitachi) et al., "ICAS: An Integrated Computer Aided Software Engineering System," **IEEE Digest of Papers-- Spring '83 COMPCON** (IEEE, 1983), pp. 239-240; Kataoka Masanori (Hitachi Software Works) et al., "Hyojun kozo ni motozuku keito-teki sofutouea sekkei ho" (Software Specification and Design Method Based on Integrated View of Structure Standardization), **Joho shori** (Information processing), Vol. 25, No. 11, November 1984, pp. 1220-1227.

57. Werner L. Frank, **Critical Issues in Software** (New York: John Wiley & Sons, 1983), pp. 34-36; Brooks, pp. 120-123; Shooman, pp. 483-489.

58. Brooks, p. 16.

59. Brooks, p. 31.

60. Again, Japanese companies did not all follow the same approach. NEC usually was able to follow Brooks' advice but also tries to add its best people to projects that are delayed. Fujitsu relies on overtime rather than adding people. Interviews with Sakata (NBC/Hitachi), 9/10/85, and Shibata (Hitachi), 9/19/85; Azuma (NEC), 9/26/85; Yoshida Tadashi (Deputy Manager of the Quality Assurance Department, Software Division/Computer Systems Group, Fujitsu), 9/24/85 and 7/31/86. According to Hitachi data, about 12% of projects have been late between 1974 and 1985, with annual rates ranging between 6.9% (1974) and 18.8% (1983) (data from Shibata, 7/23/86). As noted earlier, Hitachi Software Engineering claimed that 98% of its projects came in on time. According to Azuma, about 10% of projects late was common at NEC, although this was a personal estimate. According to Yoshida, Fujitsu reduced the number of projects its deems late (defined as received for final inspection after the scheduled date) from about 40% ca. 1978-1979 to 14.6% in 1981, 14.8% in 1982, and 12.1% in 1983 (7/31/86 interview).

61. For productivity data at Toshiba, see Yoshihiro Matsumoto (Toshiba), "A Software Factory: An Overall Approach to Software Production" (2 December 1986), forthcoming in Peter Freeman, ed., **Software Reusability** (IEEE Tutorial, 1987); Y. Matsumoto et al., "SWB System: A Software Factory," in H. Hunke, ed., **Software Engineering Environments** (Amsterdam: North-Holland, 1981), pp. 305-318; "New Toshiba Software Facility Spurs Productivity," **Toshiba Newsletter**, No. 256, November 1983, p. 1; K.H. Kim, "A Look at Japan's Development of Software Engineering Technology," **Computer**, May 1983, pp.31-33.

62. This was recently pointed out by a manager at IBM (which, as an organization, paid little attention to reusabiity), who observed that reusability "assures increasing quality over time" and "concentrates effort on new aspects of product," as well as "delivers function with less effort and error" and "abbreviates testing." See G.F. Hoffnagle and W.E. Beregi, "Automating the software development process," **IBM Systems Journal**, Vol. 24, No. 2 (1985), p. 110.

63. For a history of the development of Japanese quality control practices, see Cusumano (1985), Chapter 6. For a sampling of Japanese company discussions on quality control practices applied to software see: Yukio Mizuno (NEC), "Software Quality Improvement," **Computer**, March 1983, pp. 69-71, and "A Quantitative Approach to Software Quality and Productivity Improvement," NEC Corporation (undated manuscript); Tadashi Yoshida (Fujitsu), "Attaining Higher Quality in Software Development -- Evaluation in Practice," **Fujitsu Scientific and Technical Journal**, Vol. 21, No. 3 (July 1985), pp. 305-316; and Hashimoto Yaichiro et al. (Hitachi), "Sofutouea hinshitsu hyoka shisutemu 'SQE'" (Software Quality Estimation System `SQE'), **Hitachi hyoron**, Vol. 68, No. 5 (May 1985), pp. 55-58.

64. Several special issues of Hitachi's in-house technical journal discuss the development of these software systems and their applications. See **Hitachi hyoron**, December 1980 and May 1986 in particular. For NEC's system, see Uenohara Michiyuki (NEC) et al., "Development of Software Production and Maintenance System," **Research and Development in Japan Awarded the Okochi Memorial Prize** (Okochi Memorial Foundation, 1984), pp. 26-27; and Kanji Iwamoto (NEC), et al., "Early Experiences Regarding SDMS Introduction into Software Production Sites," **NEC Research and Development**, January 1983, p. 51.

65. Conceptualizations of the software life cycle already already appear to borrow from the model hardware development presents. For example, one of the earliest proponents of "software engineering," TRW's Barry Boehm, talks about the software life cycle as containing seven phases: (1) System Requirements, (2) Software Requirements, (3) Preliminary Design, (4) Detailed Design, (5) Coding and Debugging, (6) Testing and Pre-Operations, and (7) Operations and Maintenance. These steps correspond closely to the "phases of product development for highly engineered office equipment" (produced by IBM) described in a popular textbook on product management: (1) Proposal,

(2) Specifications, (3) Prototype, (4) Production Model, (5) Manufacturing, (6) Delivery. Assuming that "Operations and Maintenance" on the software side includes delivery, and that "Manufacturing" on the hardware side includes testing, adding "Service" to the hardware product life cycle would complete an analogy to software. See B.W. Boehm, "Software Engineering," **IEEE Transactions on Computers**, December 1976, Vol. C-25, No. 12, pp. 1126-1141; and Edgar A. Pessemier, **Product Management: Strategy and Organization**, New York, John Wiley & Sons, 1982, p. 362.

66. See R. Goldberg, "Software Engineering: An Emerging Discipline," **IBM Systems Journal**, Vol. 25, Nos. 3/4, 1986, pp. 334-353. A discussion of this shift in focus to enviroments can also be found in Horst Hunke, ed., **Software Engineering Environments** (Amsterdam: North-Holland, 1981).

67. For a discussion of strategic management concepts see Arnoldo C. Hax and Nicolas S. Majluf, **Strategic Management: An Integrative Perspective** (Englewood Cliffs, N.J., Prentice-Hall, 1984), especially pp. 72-107.

68. In terms of computing performance per cost, mainframes produced by Hitachi, Fujitsu, and NEC for several years have all been ranked equivalent or superior to IBM machines. See, for example, Dale F. Farmer, "IBM-Compatible Giants," **Datamation**, December 1981, pp. 92-104; "2 New Computers from I.B.M. Rival," **The New York Times**, 12 March, 1985, p. D5; **Nikkei Computer**, 4 March 1985, pp. 49-50 (Japanese). For supercomputers, where NEC may very well be taking the lead, see Raul Mendez and Steve Orszag, "The Japanese Supercomputer Challenge," **Datamation**, 15 May 1984, pp. 113-119.

4534 099

Date Due

FEB 20 1989
JAN 11 19[illegible]
JAN 17 [illegible]
MAY 03 1992
FEB 28 2003

Lib-26-67

MIT LIBRARIES
3 9080 005 133 928